AD-A046 573 TRW DEFENSE AND SPACE SYSTEMS GROUP HUNTSVILLE ALA F/G 9/2

REVS MAINTENANCE MANUAL. SREP FINAL REPORT. VOLUME III.(U)

AUG 77 W E BENOIT, P N BERGSTRESSER DASG60-75-C-0022

UNCLASSIFIED TRW-27332-6921-026-VOL-3 NL

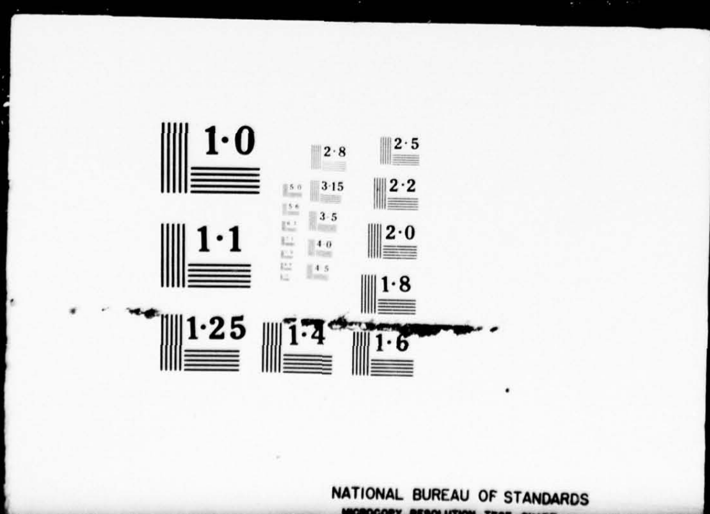
1 OF 5

ADA

046573

1 OF 5

ADA
046573




1.0 INTRODUCTION

27332-6921-026

2

B.S.

# REVS MAINTENANCE MANUAL

## SREP FINAL REPORT - VOLUME III

CDRL C005

1 AUGUST 1977



Prepared For
BALLISTIC MISSILE DEFENSE
ADVANCED TECHNOLOGY CENTER
DASG60-75-C-0022





TRW
DEFENSE AND SPACE SYSTEMS GROUP
HUNTSVILLE, ALABAMA

TRW

TITLE: REVS MAINTENANCE MANUAL DATE: 30 SEPTEMBER 1977

DOCUMENT NO: 27332-6921-026

REVISION: A

REASON FOR CHANGE:

This revision documents the ARC CDC 7600 installation of REVS.

INSTRUCTIONS:

To update this manual, make the following changes.

AFFECTED PAGES:

iii, v, ix
1-1
2-1, 2-2, 2-3
2-4 (add)
3-32
6-1
7-1, 7-18, 7-19
9-2
A-1
A-2 (add)
B-1 through B-25 (add)



SYSTEMS GROUP OF TRW INC.
ARMY SUPPORT FACILITY • 7702 GOVERNORS DRIVE WEST, HUNTSVILLE, ALABAMA 35805 (205) 837-2400

# RECORD OF REVISIONS

| REVISION | DATE | DESCRIPTION |
|---|---|---|
| A | 9/30/77 | Documents the ARC CDC 7600 installation of REVS. |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>CDRL C005 (Volume III) | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER<br>9 |
| 4. TITLE (and Subtitle)<br>6 REVS Maintenance Manual.<br>(SREP Final Report, Volume III). | | 5. TYPE OF REPORT & PERIOD COVERED<br>Final Technical Report. |
| | 14 | 6. PERFORMING ORG. REPORT NUMBER<br>TRW-27332-6921-026-VOL-3 |
| 7. AUTHOR(s)<br>10 W. E./Benoit, et al P. N./Bergstresser<br>L. J./Gunther, W. G./Heckler G. C./Hitt | | 8. CONTRACT OR GRANT NUMBER(s)<br>15 DASG60-75-C-0022 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>TRW Defense and Space Systems Group<br>7702 Governors Drive, West<br>Huntsville, Alabama 35805 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS<br>6.33.04.A |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>Ballistic Missile Defense Advanced Technology<br>Center, P. O. Box 1500, Huntsville, AL 35807<br>ATTN: ATC-P | 11 | 12. REPORT DATE<br>1 August 1977<br>13. NUMBER OF PAGES<br>422 |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)<br>12 455p. | | 15. SECURITY CLASS. (of this report)<br>UNCLASSIFIED<br>15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Cleared for public release - distribution unlimited.
Reference BMDSC-CRS letter dated 8 March 1977.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Requirements Engineering and Validation System
Requirements Specification Language
Automated Simulation Generation
Automated Documentation
Language Processors
Software Requirements Engineering

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document presents maintenance material for the Requirements Engineering and Validation System, a software system to support the generation, validation, and documentation of software requirements.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

407 674

LB

27332-6921-026

# REVS MAINTENANCE MANUAL

## SREP FINAL REPORT - VOLUME III

CDRL C005

1 AUGUST 1977

CLEARED FOR PUBLIC RELEASE - DISTRIBUTION UNLIMITED. REFERENCE BMDSC-CRS LETTER DATED 8 MARCH 1977.

THE FINDINGS OF THIS REPORT ARE NOT TO BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION.

Prepared For

BALLISTIC MISSILE DEFENSE
ADVANCED TECHNOLOGY CENTER

DASG60-75-C-0022

TRW

DEFENSE AND SPACE SYSTEMS GROUP
Huntsville, Alabama

27332-6921-026

REVS MAINTENANCE MANUAL

SREP FINAL REPORT - VOLUME III

CDRL C005 1 AUGUST 1977

Principal Authors:

W. E. Benoit
P. N. Bergstresser
L. J. Gunther
W. G. Heckler
G. C. Hitt
D. E. McQueen (AIC)
R. W. Smith

Approved By:

L. R. Marker

L. R. Marker, Manager
Software Requirements
Engineering Methodology Program

M. E. Dyer

M. E. Dyer, Manager
SREP Software and Language
Development

James E. Long

James E. Long, Manager
Huntsville Facility

Prepared For

BALLISTIC MISSILE DEFENSE
ADVANCED TECHNOLOGY CENTER

DASG60-75-C-0022



TRW.

DEFENSE AND SPACE SYSTEMS GROUP
Huntsville, Alabama

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)



LIST OF TABLES

# 1.0 INTRODUCTION

The Requirements Engineering and Validation System (REVS) is a software system designed to support the development and validation of software requirements. REVS was developed for the Ballistic Missile Defense Advanced Technology Center (BMDATC) under the Software Requirements Engineering Program, a research program concerned with the development of a systematic approach to the development of complete and validated computational requirements specifications and consists of: concepts and structured techniques for decomposition of requirements, the Requirements Statement Language (RSL), REVS, and the procedures for their application.

## 1.1 PURPOSE OF MANUAL

This maintenance manual provides information necessary to maintain the Requirements Engineering and Validation System (REVS) installed on both the Texas Instruments Advanced Scientific Computer (ASC) and the CDC 7600 Computer located at the BMDATC Advanced Research Center (ARC). The operation of each REVS function is documented to provide a clear understanding of its processing and functional organization. This material updates the information provided in the REVS Software Design Document [1].

REVS is implemented and maintained in the Process Design Language (PDL 2) [2] on the TI-ASC while on the CDC 7600 it is implemented and maintained using the PASCAL compiler and other SCOPE 2 System support software as outlined in Section 2. The REVS source code contains documentation of the operation of the REVS procedures obtainable from the Process Design System (PDS 2). This same information is retained on the REVS program library file in the form of a single deck (REVSDOC) and can be obtained via the CDC UPDATE [22] source manager. This manual does not repeat that information. Instead it is intended to provide maintenance personnel with an understanding of each operation REVS performs and to guide the programmer to the REVS procedures which perform these operations. Thus this manual should be used in conjunction with the source code documentation.

## 1.2 PURPOSE OF SOFTWARE

The Requirements Engineering and Validation System (REVS) provides the capability to maintain, analyze, simulate, and document software requirements

stated in the Requirements Statement Language (RSL) [3]. RSL and REVS were designed to meet the needs of Ballistic Missile Defense (BMD) systems and other large real-time systems with imbedded software, and to provide a degree of precision, automation, and confidence in software requirements development unattainable by conventional means.

# 2.0 ENVIRONMENT/SYSTEM DESCRIPTION

The Requirements Engineering and Validation System (REVS) operates on both the Texas Instruments Advanced Scientific Computer (ASC) and the Control Data 7600 Computer at the BMDATC Advanced Research Center (ARC) in Huntsville, Alabama. The following subsections identify the system software and hardware environment for the two computer systems at the ARC in which REVS operates.

## 2.1 SYSTEM OVERVIEW

REVS has been designed to operate in both on-line and off-line mode under control of the standard operating systems on both the TI-ASC and CDC 7600 to allow user selection of all REVS functions (except REVS Executive) in any order and any number of times during a single REVS execution. Simulator load modules are dynamically built under REVS control, not requiring the REVS user to use the Job Statement/Control Language. Thus, although REVS utilizes separately configuration controlled software, this is done in a manner which is transparent to the user and does not require special modification of the externally maintained software. All other software used by REVS is under the direct control of the REVS Executive.

## 2.2 ASC INTERFACING SOFTWARE

REVS operates under control of the standard TI-ASC General Purpose Operating System (GPOS), explicitly using the following components during installation or execution:

- Job Statement Language (JSL) [4]
- FILECOPY Utility [5]
- FORTRAN Compiler (NX) [6]
- Linkage Editor [7]
- System Object Library
- Supervisor Service Calls [8]
- Source Management System (SMS) [9]

- Card Image File Editor (CIFER) [10]
- Partitioned Direct Secondary Access Method (PDSAM) Utilities [11]

In addition to the standard ASC system software, REVS uses the TI-Huntsville developed Process Design System (PDS 2) [2] and the following components in particular:

- Source Library Management System (SLMS)
- Configuration Processor
- Process Design Language Compiler (PDL 2)
- Object Module Processor
- Overlay Processor
- Object Library Utilities
- PDS Macros

REVS also utilizes several libraries on the ASC that have been installed by the ARC contractor for local use:

- Off-line Plotter Library (CALCOMP) [12]
- On-line Color Graphics System (ANAGRAPH) [13, 14]

## 2.3 CDC INTERFACING SOFTWARE

REVS operates under control of the SCOPE 2.1 operating system on the CDC 7600 computer system located at the ARC. The following system components are explicitly used during REVS installation or execution.

- JOB CONTROL LANGUAGE FOR SCOPE 2 [20]
- FORTRAN COMPILER (FTN) [21]
- UPDATE [22]
- PASCAL Compiler [25]
- LOADER [23]
- Off-line Plotter Library (CALCOMP) [12]
- On-line Color Graphics (ANAGRAPH) [13,14]

- LIBEDT (Library Manager) [20]
- COMPASS (CDC Assembler) [24]

## 2.4 ASC COMPUTER AND INTERFACING HARDWARE

REVS provides for the use of all ARC ASC hardware facilities available through the operating system:

- o Central Processor
- o Central Memory
- o Memory Extension
- o Disks (Head per track)
- o Tapes (7 and 9 track)
- o Card Reader and Punch
- o Line Printer

In addition to the standard ASC configuration, there are other hardware systems installed at the ARC with which REVS interfaces and utilizes:

- o Off-line Paper Plotter (CALCOMP) via 7 track tape
- o On-line Color Graphics System (ANAGRAPH)

## 2.5 CDC-7600 COMPUTER AND INTERFACING HARDWARE

REVS provides for use of the following CDC hardware facilities available through the operating system.

- Small Core Memory (SCM)
- Large Core Memory (LCM)
- Central Processor
- Disks
- Tapes (7 and 9 track)
- Card Reader and Punch
- Line Printer

As on the ASC, REVS on the CDC-7600 also interfaces with an utilizes the following hardware systems installed at the ARC.

- Off-line Paper Plotter (CALCOMP)
- On-line Color Graphics (ANAGRAPH)

# 3.0 COMPUTER PROGRAM DESCRIPTION - REVS

The Requirements Engineering and Validation System (REVS) maintains, analyzes, simulates, and documents software requirements stated in the Requirements Statement Language (RSL). The REVS software is organized by function into the following components:

- REVS Executive
- RSL Translation (RSL, RSLXTND)
- Interactive R-Net Generation (RNETGEN)
- Requirements Analysis and Data Extraction (RADX)
- Simulator Generation (SIMGEN)
- Simulator Execution (SIMXQT)
- Simulation Data Analysis (SIMDA).

REVS is controlled by the user through the REVS Control Language (RCL). RSL, RCL, and the operational instructions for each of the REVS functions are documented in the REVS Users Manual [3]. These functions are briefly described below and are documented in the remainder of this section. The Simulator Program generated by REVS is documented in Section 4. The REVS Users Manual [3] presents a complete list of messages which can be generated by any of the REVS functions.

## REVS Executive

REVS is controlled at the highest level by the REVS Executive. It processes the Executive portions of the REVS Control Language (RCL) to invoke the various REVS functions and to select Executive options. The REVS functions, with the exception of RNETGEN, interface with the user through the Executive input and output procedures. The functions access the requirements data base, the Abstract System Semantic Model (ASSM), using the ASSM Access utilities of the Executive.

## RSL Translation

RSL Translation parses RSL statements, performs syntax and semantics checks and enters the requirements into the ASSM or modifies the ASSM as directed by the RSL. The RSL translation function supports both the entry/modification of requirements in RSL and the entry/modification of extensions to RSL.

## Interactive R-Net Generation

RNETGEN allows the user to interactively create, retrieve, and modify R-Nets in a graphical form from the Anagraph terminal. The R-Nets and graphics information are maintained in the ASSM. RNETGEN also permits the user to develop a graphical representation of an R-Net previously defined through RSL.

## Requirements Analysis and Data Extraction

RADX performs analysis on the requirements stored in the ASSM and provides the user with diagnostics concerning consistency and completeness of the requirements. RADX also contains a generalized query system which permits the user to selectively extract information from the ASSM and output it in RSL.

## Simulation Generation

SIMGEN translates the R-Nets, data and simulator model descriptions and their relationships established in the ASSM into Process Design Language (PDL 2) components, consolidates these with REVS provided simulator utilities and an externally provided driver, and compiles and link edits the procedural code into an executable simulator.

## Simulation Execution

Simulator run-time parameters are processed by SIMXQT, in preparation for execution of a REVS generated simulator.

## Simulation Data Analysis

Simulation post-processor run-time control parameters are processed by SIMDA. A parameter file to be read by the Simulator Post-Processor Program is also constructed by SIMDA.

## 3.1 REVS EXECUTIVE (REVS)

Description

The REVS Executive function establishes the initial conditions needed for the execution of REVS, changes state from executive to function and back to selectively execute REVS functions as directed by RCL Executive Commands (REVS-EXEC-RCL), changes mode from off-line to on-line and back, and terminates execution when requested. All input statements from XXREVSIN are routed to XXREVSOUT before performing the specified operation. Unrecognizable statements are merely flushed with a diagnostic while valid executive state statements are accompanied by action messages. Although the Executive passes control to functions as specified by RCL, it retains ultimate supervision and terminates functions and REVS executions in an orderly manner when run time errors require it.

Input

| | | |
|---|---|---|
| REVS-EXEC-RCL | - | Executive state RCL only |

Processing

REVS Executive processing is shown in the flow diagram of Figure 3-1. The following comments refer to processing steps in the flow diagram.

| | | |
|---|---|---|
| [1] | - | All Executive variables and files are set as well as global variables and files required for inter-function use (e.g., ASSM Access and Calcomp plot subfunctions). |
| [4] | - | The Executive receives only executive RCL statements which are limited to executive state use only. Other executive RCL statements which are immediate are processed within XXREVSIN. |
| [5] | - | XXREVSOUT is sent a demarcation line before and after function execution to clearly identify which output is from the Executive and which is from the function. If a graphics console synchronization error occurs following RNETGEN execution, the Executive notes this on XXREVSOUT and recovers. |

[6] - There is no limit to the number of mode changes unless conditioned by the user with a GO ONLY statement which causes the next GO statement to be interpreted as a STOP statement. An appropriate message is issued as for all other executive actions.

[7] - A mode change to on-line causes one Anagraph console to be reserved by the Executive with an initial display of the TRW/SREP logo accompanied by the user identification. A mode change to off-line causes the same TRW/SREP display followed by release of the console.

[9] - The Executive never runs out of input before encountering the STOP statement just as functions never run out of input before encountering the FEND statement, thereby obviating the need for an independent end-of-file test.

[11] - The user can specify that the job is to be stopped with the STOP JOB statement. This kind of abort suppresses tape saves or simulator builds/executions.

Procedure References

The following list correlates the functional processing steps shown in Figure 3-1 with the REVS procedures in which the processing is performed.

[1] - XXUINIT

[2-9] - XXREVS

[10] - XXUTERM

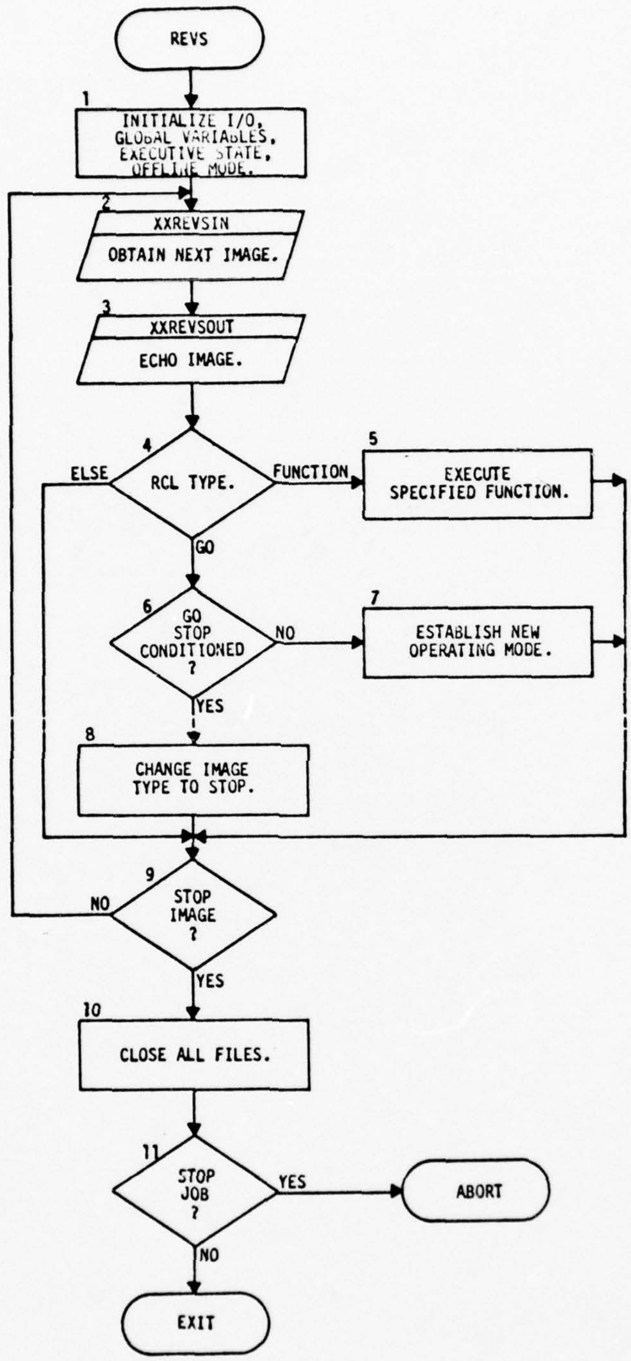



Figure 3-1 REVS Executive (REVS)

### 3.1.1 REVS Input (XXREVSIN)

Description

XXREVSIN is the Executive component which provides the standardized input interface for the Executive and all REVS functions. XXREVSIN supports the standard off-line (batch) input file REVSIN, as well as additional user specified files, and the on-line color graphics console. Although on-line input images are limited to 72 characters in length, XXREVSIN will support off-line files with records up to 132 characters in length. However, the normal size is 80. On each use of XXREVSIN the next image is returned from the current input file, which may be the standard REVSIN file, an alternate file, or the on-line console, as previously specified by REVS-EXEC-RCL. XXREVSIN logs all executive RCL unconditionally and logs function RCL and RSL if specified by the user with a LOG ALL statement. The file source of the input is shown on the log and executive RCL is time stamped.

XXREVSIN buffers (temporarily saves) some executive RCL statements when a state change is implied, and generates an appropriate implied executive RCL statement. A subsequent call will then obtain the buffered image.

XXREVSIN contains the executive RCL translator since some RCL is immediate in nature and is executed immediately upon detection. The translated statement is retained for use by the REVS Executive in those cases where executive state is required for processing. Only semantically and syntactically legal executive RCL statements are recognized by this translator, and all other input images are considered to be function RCL or RSL.

When in the on-line mode, the status display is updated whenever executive RCL is processed so that the on-line user always has a current display of the executive state variables.

Input

REVSIN FILES - Standard input, alternate addfiles, or color console keyboard.

Output

IMAGE - The next input image available to the caller, with blank fill and computed length.

| | | |
|---|---|---|
| FEND | - | Function end of data flag. |
| STATDISPLAY | - | On-line status display. |

Processing

XXREVSIN processing is shown in the flow diagram of Figure 3-2. The following comments refer to processing steps in the flow diagram.

| | | |
|---|---|---|
| [1] | - | No function is allowed to read past the FEND image and is aborted if it tries. |
| [4] | - | Input images that require a state change from function to executive are buffered and an implicit FEND is generated to synthesize what the user should have provided. |
| [5] | - | The next image is obtained from the input file REVSIN, the on-line console keyboard, or an alternate file as determined by the specification of mode and addfile by the user. |
| [7] | - | The free form image is translated to a coded matrix for use by this module and the executive module. |
| [10] | - | This step is currently superfluous as the RCL translator now includes this logic for efficiency of translation. |
| [12] | - | Logging of function RCL is only performed if explicitly requested by the user. |
| [16] | - | Executive RCL which requires the executive state causes image buffering when encountered in the function state. |
| [21] | - | Executive RCL which does not require executive state is processed within XXREVSIN. |
| [22] | - | The return status is set based on whether or not the transmitted image is a FEND. |

Procedure References

The following list correlates the functional processing steps shown in Figure 3-2 with the REVS procedures in which the processing is performed.

| | | |
|---|---|---|
| [2] | - | XXREVSLOG |
| [3] | - | XXHALT |
| [5] | - | XXGETIMAGE |
| [7] | - | XXCLASSIFY |
| [13] | - | XXREVSLOG |
| [15] | - | XXREVSLOG |
| [19] | - | XXREVSLOG |
| [21] | - | XXPERFORMANCE |

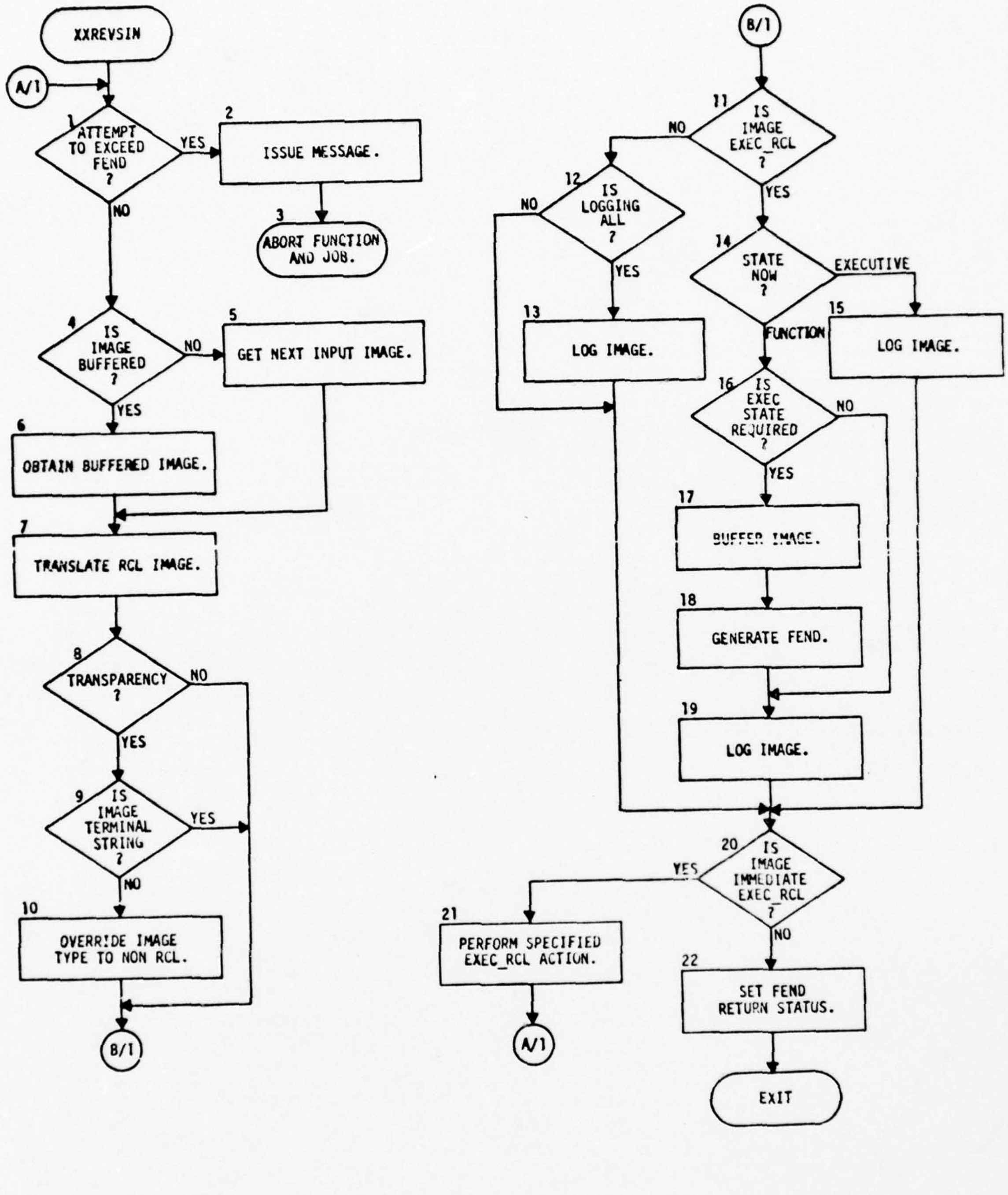



Figure 3-2 REVS Input (XXREVSIN)

### 3.1.2 REVS Output (XXREVSOUT)

Description

XXREVSOUT provides the standard output interface for the REVS Executive and functions which causes print line images to be output to the off-line printer and/or to the on-line console. On-line operations allow paging at a rate determined by the user by requiring user stimulus when he is finished with a page. A page wrapping technique is used which allows the user to view a complete page at all times. Screen conditioning is required upon first use of the console, or after dedicated use of the screen area by a function such as Interactive R-Net Generation. REVS functions are, therefore, insulated from the details of output routing which is user controlled. The user stimulus (via the trackball) will specify to continue or to interrupt the output flow. This status will be available to the function to enable interruption of lengthy outputs and to direct a return to the input process to determine the user's next request. The interrupt status will be reset on the next call to XXREVSIN. In addition, the user can change the output routing of off-line at that point if he so desires. That would not retroactively affect previously displayed output.

Input

IMAGE - The variable length image to be printed/displayed according to current routing. The image has a length component.

SPACING - Spacing controls for off-line printing.

Output

IMAGE - A line is printed and/or displayed on the console.

INTERRUPT - A signal to the calling program if the user indicates an interruption is desired.

Processing

XXREVSOUT processing is shown in the flow diagram of Figure 3-3. The following comments refer to processing steps in the flow diagram.

[1] - On-line routing is legal but ignored if not in on-line mode.

[3] - The user controls the speed of page wrapping by this pause for him to signal when ready.

[4] - Previous page is not cleared to allow maximum visibility of previous lines.

[6] - User can change output routing dynamically without explicit RCL.

[9] - User can request function interrupt. Function response is not enforced. Intended for large printing functions (RADX is responsive).

[13] - Line below current one is blanked to highlight page bottom.

[14] - Current line is always underscored in red to highlight it.

[15] - On-line function RNETGEN can reserve the normal XXREVSOUT page area for its use in which case only one line is left to use.

[17] - Off-line routing is optional.

[18, 21] - Automatic page skipping is performed on XXREVSOUT.

Procedure References

The following list correlates the functional processing shown in Figure 3-3 with the REVS procedures in which the processing is performed.

[6] - XXPERFORMRCL

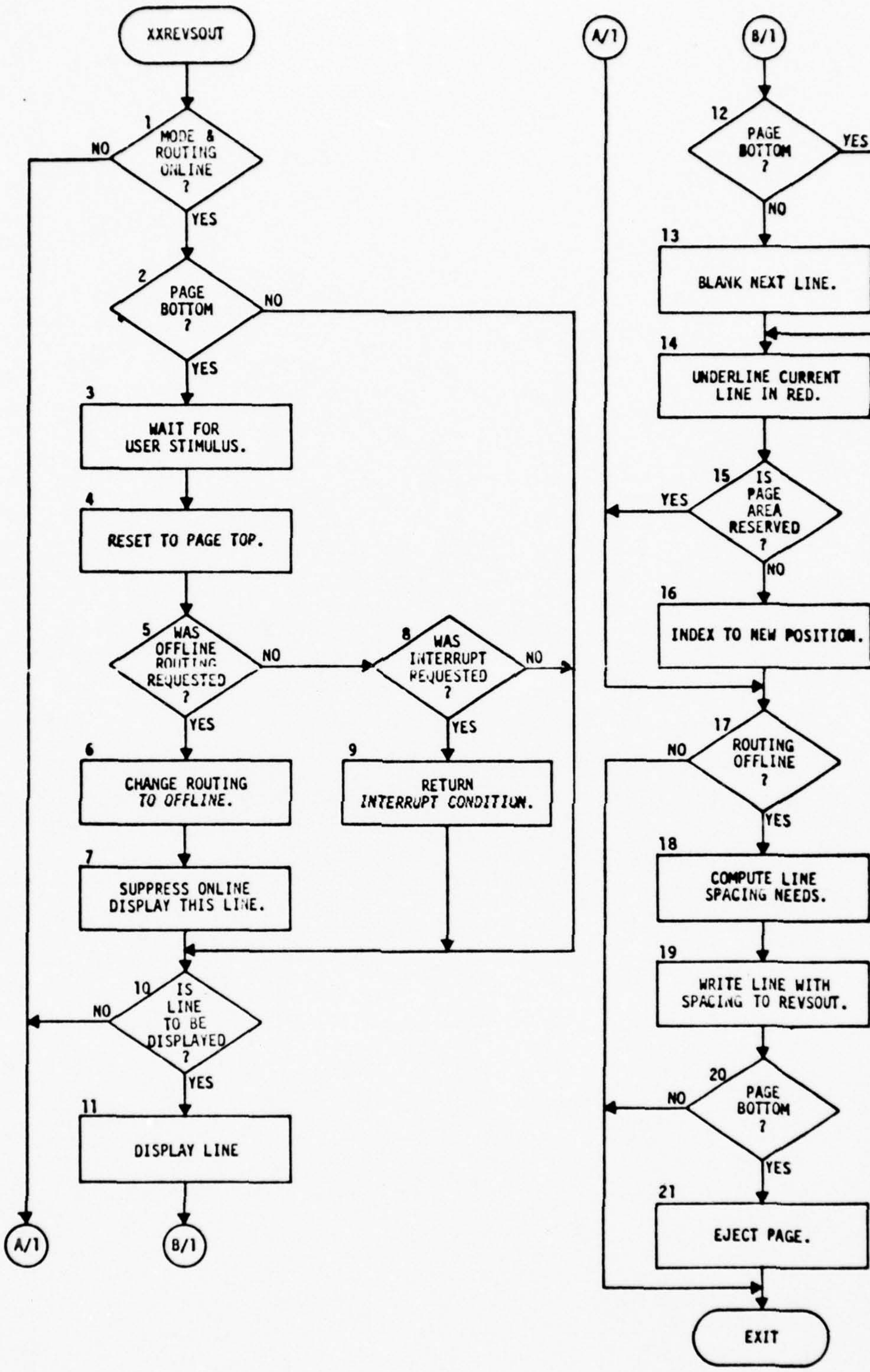



Figure 3-3 REVS Output (XXREVSOUT)

### 3.1.3 ASSM Access

Use of the core Data Base Control System (DBCS) [15] software to interface directly with the ASSM at the very lowest level would require more knowledge of the data base system than the REVS tools designer requires or cares to know. Therefore, a higher level interface to the ASSM, via the aforementioned DBCS software, has been implemented which more conveniently meets the needs imposed by the various REVS functions described in succeeding subsections in Section 3.

The ASSM interface software can be grouped into four major categories: ASSM Storage, ASSM Retrieval, ASSM Deletion, and ASSM Utilities. A list and brief description of the ASSM Access procedures is presented below by category.

#### ASSM Storage

This group of ASSM access procedures provides for storage into the ASSM of all basic components which make up RSL in addition to some structure related components which can only be entered via the RNETGEN function.

| | | |
|---|---|---|
| AAEET | - | Enters an element type. |
| AAEE | - | Enters an element. |
| AAEPR | - | Enters a primary relationship. |
| AAECR | - | Enters a complementary relationship. |
| AAERST | - | Enters a relationship subject element type. |
| AAERQT | - | Enters a relationship object element type. |
| AAEOW | - | Enters an optional word. |
| AAEPOW | - | Associates an optional word to a primary relationship. |
| AAECOW | - | Associates an optional word to a complementary relationship. |
| AAEA | - | Enters an attribute. |
| AAEAAT | - | Enters an attribute applicable element type. |

| | | |
|---|---|---|
| AAEALV | - | Enters an attribute legal value. |
| AAENPET | - | Enters legal net/path element type. |
| AAERW | - | Enters a reserved keyword. |
| AAECS | - | Enters a comment segment. |
| AAETS | - | Enters a text string segment. |
| AAAC | - | Associates a text string to an applicable ASSM component. |
| AAERI | - | Enters a relationship instance. |
| AAEAI | - | Enters an attribute instance. |
| AAEAWV | - | Enters an attribute instance without its value. |
| AAAAV | - | Associates an attribute value to an attribute instance in the ASSM. |
| AAENPN | - | Enters a node on a structure. |
| AAESPR | - | Enters the successor/predecessor relationship between two nodes on a structure. |
| AAENCXY | - | Enters graphic coordinates of a node. |
| AAENCOL | - | Enters color attribute of a node. |
| AAENAXY | - | Enters graphic coordinates for the arc between two nodes. |
| AAEORD | - | Enters the ordinal value of a structure branch. |
| AAECE | - | Enters the conditional expression for a structure branch. |
| AAENAE | - | Associates an element to a node in a structure. |
| AAACE | - | Associates a conditional expression to a node branch. |
| AAENBAE | - | Associates an element to a node branch in the ASSM. |
| AAEPER | - | Enters a given ASSM configuration control permission with permission level. |

ASSM Retrieval

This group of ASSM access procedures provides for retrieval of all data forms which may reside in the ASSM.

| | | |
|---|---|---|
| AARET | - | Retrieves an element type. |
| AARETE | - | Retrieves the element type of a given element. |
| AARPR | - | Retrieves a primary relationship. |
| AARCP | - | Retrieves a complementary relationship given the primary. |
| AARPC | - | Retrieves the primary relationship of a given complementary relationship. |
| AARRLS | - | Retrieves a subject element type of a given relationship. |
| AARRLO | - | Retrieves an object element type of a given relationship. |
| AARPOW | - | Retrieves the optional word of a given primary relationship. |
| AARCOW | - | Retrieves the optional word of a given complementary relationship. |
| AARA | - | Retrieves an attribute. |
| AARAET | - | Retrieves an applicable element type for a given attribute. |
| AARALV | - | Retrieves the legal value for a given attribute. |
| AARNPET | - | Retrieves a legal net/path element type. |
| AARRW | - | Retrieves a reserved keyword. |
| AARCS | - | Retrieves a comment segment. |
| AAREET | - | Retrieves an element of a given element type. |
| AARRSE | - | Retrieves a subject element of a given relationship instance. |
| AARROE | - | Retrieves an object element of a given relationship instance. |
| AARRS | - | Retrieves a relationship instance given its subject element. |

| | | |
|---|---|---|
| AARRO | - | Retrieves a relationship instance given its object element. |
| AARAAE | - | Retrieves an attribute instance given its applicable element. |
| AAREA | - | Retrieves an applicable element given an attribute instance. |
| AARAVE | - | Retrieves the attribute value for a given attribute instance. |
| AARRI | - | Retrieves a relationship instance. |
| AARAI | - | Retrieves an attribute instance. |
| AARE | - | Retrieves an element. |
| AARFN | - | Retrieves the first node of a given structure. |
| AARSN | - | Retrieves the successor node of a given node. |
| AARPN | - | Retrieves the predecessor node of a given node. |
| AARNT | - | Retrieves the type for a given node. |
| AARNCXY | - | Retrieves graphic coordinates for a given node. |
| AARNCOL | - | Retrieves the color attribute of a given node. |
| AARNAXY | - | Retrieves the graphic coordinates for the arc between two nodes. |
| AARORD | - | Retrieves the ordinal value on a node branch. |
| AARCE | - | Retrieves the conditional expression associated with a node branch. |
| AARTNODE | - | Retrieves a node in the temporary structure area. |
| AAREAN | - | Retrieves an element associated with a given node. |
| AARNAE | - | Retrieves a node associated with a given element. |
| AARNBAE | - | Retrieves a node branch associated with a given element. |

| | | |
|---|---|---|
| AAREANB | - | Retrieves an element associated with a given node branch. |
| AARTS | - | Retrieves a text string. |
| AARRST | - | Retrieves next relationship having a given legal subject type. |
| AARROT | - | Retrieves next relationship having a given legal object type. |
| AARAAET | - | Retrieves next attribute having a given legal applicable element type. |

ASSM Deletion

This group of ASSM access procedures provides for removal of all data forms which may reside in the ASSM.

| | | |
|---|---|---|
| AADET | - | Removes an element type. |
| AADPR | - | Removes a primary relationship. |
| AADCR | - | Removes a complementary relationship. |
| AADRST | - | Removes a subject element type for a given relationship. |
| AADROT | - | Removes an object element type for a given relationship. |
| AADOW | - | Removes an optional word. |
| AADPOW | - | Removes the optional word for a given primary relationship. |
| AADCOW | - | Removes the optional word for a given complementary relationship. |
| AADA | - | Removes an attribute. |
| AADAAT | - | Removes an applicable element type for a given attribute. |
| AADALV | - | Removes the legal value for a given attribute. |
| AADNET | - | Removes a legal net/path element type. |
| AADCS | - | Removes a comment segment. |
| AADE | - | Removes an element. |

| | | |
|---|---|---|
| AADRI | - | Removes a relationship instance. |
| AADAI | - | Removes an attribute instance. |
| AADN | - | Removes a node. |
| AADSPR | - | Removes the successor/predecessor relationship between two nodes. |
| AADPER | - | Removes the given ASSM configuration control permission and its permission level. |
| AADORD | - | Removes the ordinal for a given node branch. |
| AADCE | - | Removes the conditional expression associated with a given node branch. |
| AADNBAE | - | Removes the association of a given node branch to an element. |
| AADPS | - | Removes a structure. |
| AADTS | - | Removes a text string. |

ASSM Utilities

This group of ASSM access procedures provides all additional needs, for interfacing with the ASSM and also supports many of the other ASSM access procedures.

| | | |
|---|---|---|
| AAURTYP | - | Provides for changing the type of an existing element in the ASSM. |
| AAURNAM | - | Provides for changing the name of an existing element in the ASSM. |
| AAUBS | - | Performs the necessary initialization for building a structure in the ASSM. |
| AAUES | - | Performs the cleanup necessary in the ASSM upon conclusion of generating a structure. |
| AAURCSTR | - | Retrieves the character string in the ASSM associated with a given ASSM pointer. |
| AAURPTR | - | Retrieves the ASSM pointer for a given character string. |

| | | |
|---|---|---|
| AAURART | - | Retrieves the record type (ASSM) for a given record in the ASSM. |
| AAULEN | - | Computes the length of a given character string. |
| AAUPAD | - | Pads a character string with trailing blanks. |
| AAPERID | - | Sets up current permission level for the ASSM. |

## 3.2 RSL TRANSLATION (RSL, RSLXTND)

The RSL Translator is one functional component of the Requirements Engineering and Validation System (REVS). Its purpose is to translate input stated in the Requirements Statement Language (RSL) into entries in the REVS data base, the Abstract System Semantic Model (ASSM).

The RSL Translator operates in either of two modes, basic or extension, corresponding to the REVS function to be performed, RSL or RSL extension (RSLXTND). In the basic RSL mode, the RSL Translator supports the entry, deletion and modification of requirements stated in the Requirements Statement Language. In the extension mode, the RSL Translator supports the entry, deletion and modification of the definitions of RSL element-types, attributes and relationships.

The input to the RSL Translator is in the form of RSL command lists as defined in a condensed form in Section 3.2.2. The syntax for the command lists allows an arbitrary mix of RSL and RSL extension commands; the distinction between the types of commands is enforced by the semantic interpretations for the command list constructs as defined in Section 3.2.4 (i.e., a semantic error is detected if an RSL extension command is input while the translator is operating in the basic RSL mode). This combination of the two types of commands into one syntax allowed for the most efficient and effective use of the Lecarme-Bochmann Compiler Writing System [16, 17] as an aid in the development of the RSL Translator.

### 3.2.1 Overall Structure of the RSL Translator

The RSL Translator is a procedure generated by the Lecarme-Bochmann Compiler Writing System (L-B CWS). As such, it has the general structure of L-B CWS generated compilers as described in the Compiler Writing System User's Manual [17]. The discussion below is summarized from that source. The actual usage of the Compiler Writing System and its inputs are described in Section 7.3.1.

The translator consists of four functional segments: syntactic analyzer, lexical analyzer, semantic analyzer, and error treatment. The syntactic analyzer is the central function of the translator; i.e., it controls the operation of the other parts. Each time the syntactic analyzer needs another syntactic unit, it calls the lexical analyzer. Each time it finds that a part of the source text corresponds to the application of a particular part of the grammar, it calls a procedure in the semantic analyzer to perform the appropriate action. In addition, error treament routines exist to recover from the recognized error so that the translation may continue. The error treatment routines may be called by any function of the translator. In turn, they may call the lexical or syntactic analyzers in their attempt to recover from the error. The resulting functional structure of the translator is given in Figure 3-4.

The lexical and syntactic analyzers are supplied as standard source code by the Compiler Writing System. No significant changes have been made to the syntax analyzer (parser) and only minor changes have been incorporated into the lexical analyzer to support the translation of RSL. The Compiler Writing System also provided the framework for the treatment of errors; a framework which was tailored for use by the RSL Translator. The semantic analyzer, in contrast, is almost totally dependent on the language to be translated. As such, the bulk of the significant aspects of the RSL Translator is concentrated on the semantic actions which the translator takes, and the semantic errors which it detects.

Succeeding subsections describe the general approach used in each of the functions of the translator. These descriptions, when combined with the skeleton specification of the compilers generated by the Compiler Writing System, constitute the functional design of the RSL Translator.

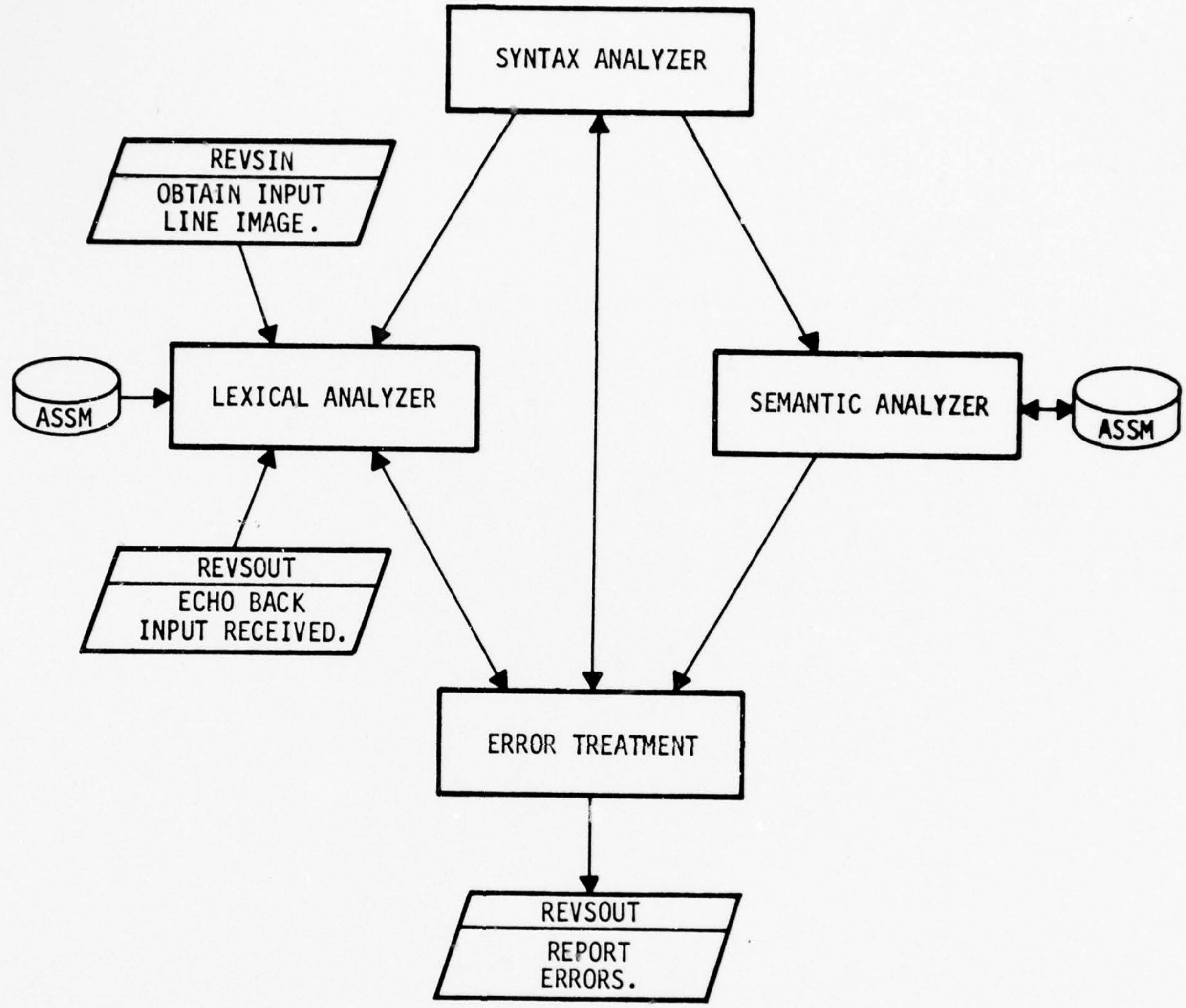



Figure 3-4 Structure of the RSL Translator

### 3.2.2 The Syntactic Analyzer

To facilitate error recovery, the RSL input is processed on a sentence-by-sentence basis, rather than on a section-by-section basis. Combination of sentences into sections is done semantically. Also, periods are considered by the parser as separators between sentences rather than a part of the sentence. This enables error messages to be issued as soon as the period is read, rather than waiting until part of the succeeding sentence has been processed.

#### Syntactic Treatment of Names (Identifiers)

There are essentially two classes of names recognized by the parser: (1) previously defined attribute, relation, and element-type-names; (2) previously defined element-names, and value-names and new (not previously defined) names. Names of class (1) are treated by the parser as separate syntactic entities, and have a semantic attribute which is a pointer to the appropriate entry in the ASSM. Names of class (2) are syntactically identical, but can be distinguished by their semantic attributes. (See Section 3.2.4, "Semantic Treatment of Names".)

#### RSL Syntax (Condensed)

Table 3.1 contains a description of the syntax of RSL in condensed form more suitable for reading by humans than the full version which is input to the L-B CWS. (See Section 7.3.1.) For each syntax production, this table also identifies the page in Section 3.2.4 on which the semantics for the production are documented.

## Table 3.1 Condensed RSL Syntax

<table>
<tr><th>RSL PRODUCTION RULE</th><th>PAGE NO.</th></tr>
<tr><td><command list>::=</td><td></td></tr>
<tr><td>{<command>}$_1^n$ end-of-file</td><td>3-33</td></tr>
<tr><td><command>::=</td><td>3-34</td></tr>
<tr><td>[<section keyword>] <definition></td><td>3-35</td></tr>
<tr><td>| RENAME element-name AS new-name [comment].</td><td>3-39</td></tr>
<tr><td>| RETYPE element-name AS element-type-name.</td><td>3-40</td></tr>
<tr><td>| <error-level command></td><td>3-41</td></tr>
<tr><td>| <extension-control command></td><td>3-42</td></tr>
<tr><td><extension-control command>::=</td><td></td></tr>
<tr><td>IDENTIFICATION name.</td><td>3-35</td></tr>
<tr><td>| EXTENSION_PERMISSION name.</td><td>3-36</td></tr>
<tr><td>| CONTROL_PERMISSION name.</td><td>3-37</td></tr>
<tr><td>| RESCIND PERMISSION name.</td><td>3-38</td></tr>
<tr><td><error-level command>::=</td><td></td></tr>
<tr><td>ERROR LEVEL integer.</td><td>3-42</td></tr>
<tr><td><section keyword>::=</td><td>3-43</td></tr>
<tr><td>DEFINE</td><td></td></tr>
<tr><td>| ADD</td><td></td></tr>
<tr><td>| DELETE</td><td></td></tr>
<tr><td>| MODIFY</td><td></td></tr>
<tr><td><definition>::=</td><td>3-44</td></tr>
<tr><td><element-type definition></td><td></td></tr>
<tr><td>| <attribute definition></td><td></td></tr>
<tr><td>| <relation definition></td><td></td></tr>
<tr><td>| <element definition></td><td></td></tr>
<tr><td><element-type definition>::=</td><td></td></tr>
<tr><td><element-type-definition header></td><td>3-45</td></tr>
<tr><td>{[<sentence keyword>] <element-type-definition sentence>}$_0^n$</td><td></td></tr>
<tr><td><element-type-definition header>::=</td><td>3-46</td></tr>
<tr><td>ELEMENT_TYPE element-type-name [comment].</td><td>3-47</td></tr>
<tr><td><element-type-definition sentence>::=</td><td>3-48</td></tr>
<tr><td><error-level command></td><td></td></tr>
<tr><td>| STRUCTURE APPLICABILITY {NET, PATH}$_1^1$.</td><td></td></tr>
<tr><td><sentence keyword>::=</td><td>3-49</td></tr>
<tr><td>INSERT</td><td></td></tr>
<tr><td>| REMOVE</td><td></td></tr>
<tr><td><attribute definition>::=</td><td>3-50</td></tr>
<tr><td><attribute-definition header></td><td></td></tr>
<tr><td>{[<sentence keyword>] <attribute-definition sentence>}$_0^n$</td><td></td></tr>
<tr><td><attribute-definition header>::=</td><td>3-51</td></tr>
<tr><td>ATTRIBUTE attribute-name [comment].</td><td>3-52</td></tr>
<tr><td><attribute-definition sentence>::=</td><td>3-53</td></tr>
<tr><td><error-level command></td><td></td></tr>
<tr><td>| <applicable types></td><td></td></tr>
<tr><td>| VALUE value-name [comment].</td><td></td></tr>
</table>

Table 3.1 Condensed RSL Syntax (Continued)

<table>
<tr><th>RSL PRODUCTION RULE</th><th>PAGE NO.</th></tr>
<tr><td><applicable types>::=<br>APPLICABLE [ELEMENT_TYPE] <element types>.</td><td>3-54</td></tr>
<tr><td><element types>::=<br>ALL<br>| [ALL EXCEPT] $\{$element-type-name$\}_1^n$</td><td>3-54</td></tr>
<tr><td><relation definition>::=<br><relation-definition header><br>$\{$[<sentence keyword>] <relation-definition sentence>$\}_0^n$</td><td>3-55</td></tr>
<tr><td><relation-definition header>::=<br>$\left\{\begin{matrix}\text{RELATION}\\\text{RELATIONSHIP}\end{matrix}\right\}_1^1$ <relation id> [comment].</td><td>3-56<br>3-57</td></tr>
<tr><td><relation id>::=<br>relation-name [(string)]</td><td>3-56<br>3-57</td></tr>
<tr><td><relation-definition sentence>::=<br><error-level command><br>| <complementary part><br>| <subject part><br>| <object part></td><td>3-58</td></tr>
<tr><td><complementary part>::=<br>COMPLEMENTARY $\left\{\begin{matrix}\text{RELATION}\\\text{RELATIONSHIP}\end{matrix}\right\}_1^1$ <relation id>.</td><td>3-58</td></tr>
<tr><td><subject part>::=<br>SUBJECT [ELEMENT_TYPE] [<element types>].</td><td>3-59</td></tr>
<tr><td><object part>::=<br>OBJECT [ELEMENT_TYPE] [<element types>].</td><td>3-60</td></tr>
<tr><td><element definition>::=<br><element-definition header><br>$\{$[<sentence keyword>] <element-definition sentence>$\}_0^n$</td><td>3-61</td></tr>
<tr><td><element-definition header>::=<br>element-type-name element-name [comment].</td><td>3-62<br>3-63</td></tr>
<tr><td><element-definition sentence>::=<br><error-level command><br>| <attribute declaration><br>| <relation declaration><br>| <path declaration><br>| <structure declaration></td><td>3-64<br>3-65</td></tr>
<tr><td><attribute declaration>::=<br>attribute-name $\left[\left\{\begin{matrix}\text{value-name}\\\text{number}\\\text{string}\end{matrix}\right\}_1^1 \text{[comment]}\right]$.</td><td>3-64<br>3-65</td></tr>
</table>

## Table 3.1 Condensed RSL Syntax (Continued)

<table>
<tr><th>RSL PRODUCTION RULE</th><th>PAGE NO.</th></tr>
<tr><td><relation declaration>::=<br>relation-name <object list>.</td><td>3-66<br>3-67</td></tr>
<tr><td><object list>::=<br>{[element-type-name] element-name [comment]}$_1^n$</td><td>3-66<br>3-67</td></tr>
<tr><td><path declaration>::=<br>PATH [{<validation node>}$_1^n$ END] [comment].</td><td>3-68<br>3-69</td></tr>
<tr><td><validation node>::=<br>[element-type-name] element-name [comment]</td><td>3-68<br>3-69</td></tr>
<tr><td><structure declaration>::=<br>STRUCTURE [{<node>}$_2^n$ END] [comment].</td><td>3-70<br>3-71</td></tr>
<tr><td><node>::=<br><element node><br>| <terminator><br>| <and node><br>| <or node><br>| <consider-or node><br>| <for-each node><br>| <select node></td><td>3-70<br>3-71</td></tr>
<tr><td><element node>::=<br>[element-type-name] element-name [comment]</td><td>3-72</td></tr>
<tr><td><terminator>::=<br>TERMINATE [comment]<br>| RETURN [comment]</td><td>3-73</td></tr>
<tr><td><and node>::=<br>DO [comment] <branch> {AND <branch>}$_1^n$ END</td><td>3-74</td></tr>
<tr><td><branch>::=<br>{<node>}$_1^n$</td><td>3-75</td></tr>
<tr><td><or node>::=<br>IF [comment] <conditional branch><br>{OR <conditional branch>}$_0^n$<br>OTHERWISE [<branch>] END</td><td>3-76</td></tr>
<tr><td><conditional branch>::=<br>[integer] <condition> <branch></td><td>3-77</td></tr>
<tr><td><consider-or node>::=<br><consider-data><br>| <consider-entity-class></td><td>3-78<br>3-81</td></tr>
<tr><td><consider-data>::=<br>CONSIDER [DATA] enumerated-data-name IF [comment]<br><consider-data branch><br>{OR <consider-data branch>}$_1^n$<br>END</td><td>3-78</td></tr>
</table>

Table 3.1 Condensed RSL Syntax (Continued)

<table>
<tr><th>RSL PRODUCTION RULE</th><th>PAGE NO.</th></tr>
<tr><td><consider-data branch>::=<br>(<enumeration-value-list>) <branch><br>| (<enumeration-value-list>)</td><td>3-80</td></tr>
<tr><td><enumeration-value-list>::=<br>enumeration-value-name {OR enumeration-value-name}$_0^n$</td><td>3-80</td></tr>
<tr><td><consider-entity-class>::=<br>CONSIDER [ENTITY_CLASS] entity-class-name IF [comment]<br><consider-entity branch><br>{OR <consider-entity branch>}$_1^n$<br>END</td><td>3-81</td></tr>
<tr><td><consider-entity-class branch>::=<br>(<entity-type-list>) <branch><br>| (<entity-type-list>)</td><td>3-82</td></tr>
<tr><td><entity-type-list>::=<br>entity-type-name {OR entity-type-name}$_0^n$</td><td>3-82</td></tr>
<tr><td><for-each node>::=<br>FOR EACH <for-each subject> [SUCH THAT <condition>]<br>DO [comment] <for-each body node> END</td><td>3-83</td></tr>
<tr><td><for each subject>::=<br>[FILE] file-name [RECORD]<br>| [ENTITY_TYPE] entity-type-name<br>| [ENTITY_CLASS] entity-class-name</td><td>3-84</td></tr>
<tr><td><for-each body node>::=<br>[ALPHA] alpha-name [comment]<br>| [SUBNET] subnet-name [comment]</td><td>3-85</td></tr>
<tr><td><select node>::=<br>SELECT <select subject> SUCH THAT <condition><br>[comment]</td><td>3-86</td></tr>
<tr><td><select subject>::=<br>[ENTITY_CLASS] entity-class-name<br>| [ENTITY_TYPE] entity-type-name</td><td>3-87</td></tr>
</table>

Table 3.1 Condensed RSL Syntax (Continued)

<table>
<tr><th>RSL PRODUCTION RULE</th><th>PAGE NO.</th></tr>
<tr><td><condition>::=</td><td>3-88</td></tr>
<tr><td>    (<Boolean expression>)</td><td>3-89</td></tr>
<tr><td><Boolean expression>::=</td><td></td></tr>
<tr><td>    <simple Boolean> $\{$<B op> <simple Boolean>$\}_0^n$</td><td></td></tr>
<tr><td><simple Boolean>::=</td><td></td></tr>
<tr><td>    <Boolean term> $\{$OR <Boolean term>$\}_0^n$</td><td></td></tr>
<tr><td><Boolean term>::=</td><td></td></tr>
<tr><td>    <Boolean factor> $\{$AND <Boolean factor>$\}_0^n$</td><td></td></tr>
<tr><td><Boolean factor>::=</td><td></td></tr>
<tr><td>    <Boolean> [<rel op> <Boolean>]</td><td></td></tr>
<tr><td>  | <arithmetic expression> <rel op> <arithmetic expression></td><td></td></tr>
<tr><td><Boolean>::=</td><td></td></tr>
<tr><td>    [NOT] <Boolean primary></td><td></td></tr>
<tr><td><Boolean primary>::=</td><td></td></tr>
<tr><td>    TRUE</td><td></td></tr>
<tr><td>  | FALSE</td><td></td></tr>
<tr><td>  | data-name</td><td></td></tr>
<tr><td>  | (<Boolean expression>)</td><td></td></tr>
<tr><td><arithmetic expression>::=</td><td></td></tr>
<tr><td>    [<ad op>] <arithmetic term></td><td></td></tr>
<tr><td>  | <arithemtic expression> $\{$<ad op> <arithmetic term>$\}_0^n$</td><td></td></tr>
<tr><td><arithmetic term>::=</td><td></td></tr>
<tr><td>    <arithmetic factor> $\{$<mul op> <arithmetic factor>$\}_0^n$</td><td></td></tr>
<tr><td><arithmetic factor>::=</td><td></td></tr>
<tr><td>    number</td><td></td></tr>
<tr><td>  | data-name</td><td></td></tr>
<tr><td>  | (<arithmetic expression>)</td><td></td></tr>
<tr><td><B op>::=</td><td></td></tr>
<tr><td>    EQU | XOR</td><td></td></tr>
<tr><td><rel op>::=</td><td></td></tr>
<tr><td>    = | < | > | <= | >= | <></td><td></td></tr>
<tr><td><ad op>::=</td><td></td></tr>
<tr><td>    + | -</td><td></td></tr>
<tr><td><mul op>::=</td><td></td></tr>
<tr><td>    * | / | DIV | MOD</td><td></td></tr>
</table>

### 3.2.3 The Lexical Analyzer

The function of the lexical analyzer is to scan the input stream and to generate RSL terminal symbols for the syntactic analyzer. Some semantic information is also generated for the terminals and associated with them in the form of semantic attributes. The following paragraphs identify the terminal symbols of RSL and provide a summary of the semantic attributes associated with each symbol.

End-of-file

This symbol is generated when an end-of-file indicator is returned from XXREVSIN. It has no semantic attributes.

Name

This is a sequence of up to 60 alphanumeric characters, the first one of which is alphabetic. (The underscore character is considered alphabetic.) Its semantic attributes include: (1) a flag indicating whether the name can be found in the ASSM; (2) the type of ASSM record; (3) a pointer to the ASSM record. If the name cannot be found in the ASSM, it is either a value-name or a name being defined for the first time. Names are kept in a one-position name table. When an unrecognized name or possible value-name is processed by the lexical analyzer, it is moved from working storage to the name table, and a flag is set indicating that the name table is full. When the semantic analyzer determines that the name is to be used, it moves it back to working storage and resets the flag. If the semantic analyzer determines that the name is not to be used, the flag is reset and the name in the table is discarded.

Two semantic errors can occur in the use of the name table: error number 448 (too many new (undefined) names); error number 451 (new (undefined) name was lost). These are only expected to occur as a result of syntax errors or other semantic errors.

Comment

Comments are scanned completely and stored in the ASSM (with their comment brackets) as a chain of text. The line structure of multi-line comments is preserved as follows: (1) a new comment segment is begun whenever an end-of-line indicator is returned by XXREVSIN, regardless of the number of characters scanned; (2) a single line which contains 60 or more characters is stored as two comment segments, the first containing exactly 60 characters, and the second containing the remaining characters. (In the case of a comment line containing exactly 60 characters, two segments are generated, the second of which is null.)

The semantic attribute of a comment is a pointer to the first segment of the text chain in the ASSM. This will later be associated with the appropriate ASSM record by the semantic analyzer.

String

Strings are processed in exactly the same way as comments. (The delimiting double quotes are stored along with the text.)

Integer

The binary value of an integer is returned as its semantic attribute. The character-string representing the integer is retained as a global variable, for possible use as an attribute-value. A global value is adequate for this purpose, since each attribute-value is processed as a separate sentence.

Real Number

Real numbers are defined exactly as in PASCAL. Real numbers are checked for validity, but have no semantic attributes. The character-string representing the number is retained as a global variable, for possible use as an attribute value.

Conditional Expressions

Conditional expressions are checked by the parser for syntactic validity, but are also copied into the ASSM as a chain of text segments, including the surrounding parentheses. A pointer to this string is saved as the semantic attribute of <conditional expression>. Later the semantic analyzer will associate this text chain with a node-branch of a structure.

### 3.2.4 The Semantic Analyzer

The semantics of an artificial language is the output, other than an indication of syntactic correctness, generated by the translator. In the case of RSL, the semantics consists of a series of interactions with the ASSM.

A semantic action may be taken by the RSL translator for each production in the syntactic description of RSL which was input to the L-B CWS (see Section 7.3.1). A semantic action has inputs which are the semantic attributes associated with the symbols on the right-hand side of a production. A semantic action may call the ASSM access routines to retrieve information from or enter it into the ASSM. Semantic actions may also test and set global variables which are internal to the RSL translator. The highest level at which semantic action is taken is the section. However, most information is processed at the sentence level, and the only information retained between sentences is the type of the header, the associated name in the ASSM, and the type of the <section keyword>.

Production-by-Production Semantics

This section contains a description of the semantic actions for each production of RSL, as expressed in the condensed RSL syntax shown in Table 3.1. Since each production in the condensed syntax usually corresponds to several productions in the syntactic input to the L-B CWS, there is a similar subdivision of the semantics.

The semantics for each production is expressed in two parts, normal semantic action and possible semantic errors. The tests for errors are actually performed before any semantic action is taken, but the order has been reversed in the description in order to clarify understanding of the normal case. The name or names of the software modules which perform each semantic action are indicated in parentheses following the description of the action.

Semantic errors are communicated to the user in the form of integer numbers. For each possible semantic error, the error number and the software modules which can detect the error are given in parentheses

following the description of the error. A cross-reference between the error numbers and the actual mnemonic names used in the RSL Translator source code is presented in Appendix A. A complete listing of all possible error codes and their interpretations is presented in the REVS Users Manual [3].

In most cases, a semantic error invalidates the entire sentence and a semantic error in a section header invalidates the entire section. A few semantic errors result only in a warning message, and a few cause the RSL translator to terminate abnormally (fatal errors).

The highest syntactic unit processed by the parser is the sentence, so descriptions of semantic actions above the sentence level are actually summaries of actions already taken at a lower level.

<command list>::=

$$\left\{ \text{<command>} \right\}_1^n \text{ end-of-file}$$

Semantic Action

After each command has been processed, and an end-of-file signal is received from XXREVSIN, RSL returns control to the REVS Executive.

<command>::=

[<section keyword>] <definition>

Semantic Action

The <definition> is processed in the mode corresponding to the <section keyword>. If no <section keyword> is present the default mode is DEFINE, if the name in the section header is undefined, or MODIFY, if the name in the section header was previously defined (ACTION).

<command>::=

<extension-control command>

<extension-control command>::=

IDENTIFICATION name.

Semantic Action

This command identifies the user and establishes the appropriate permission level (none, extension only, or control) in accordance with the permission level associated with the name (TTEXTCNTRL). This permission level remains in effect until superseded by another IDENTIFICATION command or by the conclusion of the function execution. The name may have a maximum of 58 significant characters to form a unique permission identifier, but is maintained separately from all other RSL names so that no conflict can arise.

Possible Semantic Errors

The RSL translator was not invoked in extension mode, i.e., via RSLXTND (500-TTEXTCNTRL).

<extension-control command>::=

EXTENSION_PERMISSION name.

Semantic Action

The name will be entered in the ASSM and extension permission associated with it (TTEXTCNTRL).

Possible Semantic Errors

1. The current permission level is not control permission (497-TTEXTCNTRL).

2. A permission is already associated with the name (498-TTEXTCNTRL).

3. No control permission exists in the ASSM (502-TTEXTCNTRL).

<extension-control command>::=

CONTROL_PERMISSION name.

Semantic Action

The name will be entered in the ASSM and control permission associated with it (TTEXTCNTRL). Note that control permission includes extension permission.

Possible Semantic Errors

1. The current permission level is not control permission (497-TTEXTCNTRL).

2. A permission is already associated with the name (498-TTEXTCNTRL).

<extension-control command>::=

RESCIND PERMISSION name.

Semantic Action

The permission associated with the name will be rescinded. This takes effect immediately unless the given name was the one used to acquire control permission on the preceding IDENTIFICATION command, in which case it will take effect at the next IDENTIFICATION statement or at conclusion of function execution.

Possible Semantic Errors

1. The current permission level is not control permission (497-TTEXTCNTRL).
2. No permission is associated with the name (499-TTEXTCNTRL).
3. The name has the only control permission in the ASSM and there is at least one outstanding extension permission (501-TTEXTCNTRL).

<command>::=

RENAME element-name AS new-name [comment].

Semantic Action

1. The name of the element will be changed (TTRENAME).

2. Any existing comment for the element will be removed (TTATTCOM).

3. The comment, if given, will be associated with the element (TTATTCOM).

Possible Semantic Errors

1. The new-name is already in use (449-ACTION).

2. The old element-name is not a valid element-name (479-ACTION).

3. The old name is an element-type-name, attribute-name, or relation-name (syntax error).

<command>::=

RETYPE element-name AS element-type-name.

Semantic Action

The element-type of the element is changed to the new element-type (TTRETYPE).

Possible Semantic Errors

1. The name is not a valid element-name (479-ACTION).

2. The element has an associated STRUCTURE and the new element-type is not SUBNET or R_NET (475-TTRETYPE).

3. The element has an associated PATH (481-TTRETYPE). (Only elements of type VALIDATION_PATH may have an associated PATH.)

4. The element is associated with a node of a structure and the new element-type has not been defined with STRUCTURE APPLICABILITY NET (427-TTRETYPE).

5. The element is associated with an OR-node or a node branch on a structure (409-TTRETYPE).

6. The element is associated with a node on a PATH and the new element-type has not been defined with STRUCTURE APPLICABILITY PATH (480-TTRETYPE).

7. The existing attributes of the element are not applicable to the new element-type (401-TTRETYPE).

8. The existing relationships of the element are not defined for the new element-type (441, 442-TTRETYPE).

9. The element-type-name is the same as the element-type of the element (487-TTRETYPE).

10. The element is associated with a FOR EACH node and the new element-type is not FILE, ENTITY_CLASS, or ENTITY_TYPE (433-TTRETYPE).

11. The element is associated with a SELECT node and the new element-type is not ENTITY_CLASS or ENTITY_TYPE (486-TTRETYPE).

12. The element is associated with a node immediately following a FOR EACH node and the element-type is not ALPHA or SUBNET (432-TTRETYPE).

13. The element is associated with a node on a STRUCTURE and the old or new element-type is OUTPUT_INTERFACE (491-TTRETYPE).

14. The element is associated with a node on a SUBNET and the new element-type is INPUT_INTERFACE (488-TTRETYPE).

15. The element is associated with a node on a STRUCTURE which follows another node and the new element-type is INPUT_INTERFACE (436-TTRETYPE).

<command>::=

<error-level command>

<error-level command>::=

ERROR LEVEL integer.

<u>Semantic Action</u>

The error message level will be set to the specified integer. The default value is 1 (ACTION).

```
<section keyword>::=
          DEFINE
        | ADD
        | DELETE
        | MODIFY
```

Semantic Action

The <section keyword> sets the mode in which the succeeding definition is to be processed. ADD and DEFINE are synonymous (ACTION).

```
<definition>::=
          <element-type definition>
      |   <attribute definition>
      |   <relation definition>
      |   <element definition>
```

Semantic Action

The <definition> will be processed in the appropriate mode (ADD, MODIFY, or DELETE), and the ASSM will be updated appropriately.

```
<element-type definition>::=
          <element-type-definition header>
             { [<sentence keyword>]
               <element-type-definition sentence> } n
                                                    0
```

Semantic Action

The header is processed according to the mode of the <section keyword> (explicit or default), and each sentence is processed according to the mode (explicit or default) of the <sentence keyword>.

Possible Semantic Errors

1. An <element-type-definition header> is followed by a sentence which is not an <element-type-definition sentence> (430-TTSNTCHK).

2. An <element-type-definition sentence> is not preceded by an <element-type-definition header> (425-TTSNTCHK).

3. An <element-type-definition sentence> appears in an <element-type definition> of mode DELETE (411-ACTION).

<element-type-definition header>::=

ELEMENT_TYPE element-type-name [comment].

Semantic Action

ADDition mode

1. A new element-type is entered into the ASSM with the given name (TTNEWELT).

2. The comment, if any, is associated with the element-type definition in the ASSM (TTATTCOM).

MODIFication mode

1. The NET/PATH indicator is changed as specified in the following sentence (TTFLAG).

2. The comment, if any, replaces the one in the ASSM (TTOLDELT).

3. If the comment is not given, any existing comment will be retained.

DELETion mode

1. The comment, if any, is deleted from the ASSM (TTDELCOM).

2. The element-type is deleted from the ASSM (TTOLDELT).

Possible Semantic Errors

ADDition mode

1. No comment is specified (201-ACTION).

2. The element-type-name is already in the ASSM (415-TTOLDELT).

MODIFication mode

1. The name is not defined in the ASSM as an element-type-name (446-TTNEWELT).

DELETion mode

1. The name is not defined in the ASSM as an element-type-name (446-TTNEWELT).

2. An element of this type exists in the ASSM (426-TTOLDELT).

3. The element-type is an applicable element-type of an attribute (403-TTOLDELT).

4. The element-type is a legal subject element-type of a relation (472-TTOLDELT).

5. The element-type is a legal object element-type of a relation (471-TTOLDELT).

<u>All modes</u>

Extension permission has not been established (496-EREXTMODE).

<element-type-definition sentence>::=

STRUCTURE APPLICABILITY $\left\{\begin{matrix}\text{NET}\\\text{PATH}\end{matrix}\right\}_1^1$.

Semantic Action

INSERT mode

The element-type is flagged as a legal NET/PATH element-type (TTFLAG).

REMOVE mode

The element-type is deleted as a legal NET/PATH element-type (TTFLAG).

Possible Semantic Errors

INSERT mode

The element-type is already flagged as a legal NET/PATH element-type (417-TTFLAG).

REMOVE mode

1. NET is specified and the element-type is not a NET legal type (457-TTFLAG).
2. PATH is specified and the element-type is not a PATH legal type (457-TTFLAG).
3. An element of the given type is used on a NET or PATH (429-TTFLAG).

<sentence keyword>::=

INSERT

| REMOVE

Semantic Action

The <sentence keyword> sets the mode in which the succeeding sentence is to be processed. If no <sentence keyword> is specified, the default mode is INSERT (ACTION).

Possible Semantic Errors

1. A REMOVE sentence appears and the <section keyword> is not MODIFY or DELETE (439-ACTION).

2. An INSERT sentence appears and the <section keyword> is DELETE (411-ACTION).

<attribute definition ::=

<attribute-definition header>

$$\left\{ \begin{array}{l} [\text{<sentence keyword>}] \\ \text{<attribute-definition sentence>} \end{array} \right\}_0^n$$

Semantic Action

The header is processed according to the mode of the <section keyword> (explicit or default), and each sentence is processed according to the mode (explicit or default) of the <sentence keyword>.

Possible Semantic Errors

1. An <attribute-definition header> is followed by a sentence which is not an <attribute-definition sentence> (405-TTSNTCHK).

2. An <attribute-definition sentence> is not preceded by an <attribute-definition header> (404-TTSNTCHK).

3. An <attribute-definition sentence> appears in an <attribute definition> of mode DELETE (411-ACTION).

<attribute-definition header>::=

ATTRIBUTE attribute-name [comment].

Semantic Action

ADDition mode

1. The new attribute-name is entered into the ASSM (TTNEWATTR).

2. The comment, if any, is associated with the attribute-name in the ASSM (TTATTCOM).

MODIFication mode

1. Applicable types and legal values are changed as specified in the following sentences.

2. The comment, if any, replaces the one in the ASSM (TTATTCOM).

3. If the comment is not given, any existing comment will be retained.

DELETion mode

1. All applicable element-types for the attribute-name will be removed from the ASSM (TTOLDATTR).

2. All legal value-names and associated comments for the attribute-name will be removed from the ASSM (TTOLDATTR, TTDELCOM).

3. The attribute-name and any associated comment will be removed from the ASSM (TTOLDATTR, TTDELCOM).

Possible Semantic Errors

ADDition mode

1. The attribute-name is already in use (449-ACTION).

2. No comment is specified (201-ACTION).

MODIFication mode

1. The name is not defined in the ASSM as an attribute-name (444-TTNEWATTR).

DELETion mode

1. The name is not defined in the ASSM as an attribute-name (444-TTNEWATTR).

2. An element exists which has this attribute (400-TTOLDATTR).

ALL modes

Extension permission has not been established (496-TTNEWATTR, TTOLDATTR).

<attribute-definition sentence>::=

<applicable types>

| VALUE value-name [comment].

Semantic Action

INSERT mode

1. The value-name is entered as a legal value for this attribute (TTVALDEF).

   (NOTE: The identifiers NAMED, NUMERIC, and TEXT are not reserved words, but have special properties when used as value-names. Cf. the semantics of <attribute declaration>.)

2. The comment, if any, is associated with the value-name (TTATTCOM).

REMOVE mode

1. The value-name is removed from the set of legal values for this attribute (TTVALDEF).

2. The associated comment, if any, is removed from the ASSM (TTDELCOM).

Possible Semantic Errors

INSERT mode

The value-name is already a legal value for this attribute (423-TTVALDEF).

REMOVE mode

1. The value-name is not a legal value for this attribute (464-TTVALDEF).

2. An element exists with this attribute and value equal to the value-name (400-TTVALDEF).

3. The legal value of this attribute is NAMED, NUMERIC, or TEXT and an element exists with the attribute (400-TTVALDEF).

4. A comment is specified (202-ACTION).

<applicable types>::=

APPLICABLE [ELEMENT_TYPE] <element types>.

<element types>::=

ALL

| [ALL EXCEPT] $\left\{ \text{element-type-name} \right\}_1^n$

Semantic Action

INSERT mode

1. The element-types on the list will be entered into the ASSM as applicable types for this attribute, unless already present (TTAPPELTYPES).

2. If ALL is specified, all currently defined element-types will be copied and entered (TTAPPELTYPES).

3. If ALL EXCEPT is specified, all currently defined element-types except those on the list will be copied and entered (TTAPPELTYPES).

REMOVE mode

1. If ALL is specified, all applicable element-types will be removed (TTAPPELTYPES).

2. If a list is specified, those element-types will be removed from the list of applicable element-types (TTAPPELTYPES).

Possible Semantic Errors

1. ALL EXCEPT is specified in REMOVE mode (452-TTAPPELTYPES).

2. An element-type is specified in REMOVE mode, and is not on the list of applicable element-types (401-TTCHKETLM).

3. An element of a type listed in REMOVE mode has this attribute (400-TTVALDEF).

4. No element-types are specified for INSERT mode (463-TTAPPELTYPES).

```
<relation definition>::=
      <relation-definition header>
          {[<sentence keyword>]
           <relation-definition sentence>}^n_0
```

Semantic Action

The header is processed according to the mode of the <section keyword> (explicit or default), and each sentence is processed according to the mode (explicit or default) of the <sentence keyword>.

Possible Semantic Errors

1. A <relation-definition header> is followed by a sentence which is not a <relation-definition sentence> (467-TTSNTCHK).
2. A <relation-definition sentence> is not preceded by a <relation-definition header> (468-TTSNTCHK).
3. A <relation-definition sentence> appears in a <relation definition> of mode DELETE (411-ACTION).

<relation-definition header>::=

$\left\{\begin{matrix}\text{RELATION}\\ \text{RELATIONSHIP}\end{matrix}\right\}_{1}^{1}$ <relation id> [comment].

<relation id>::=

relation-name [(string)].

Semantic Action

ADDition mode

1. The new relation-name is entered into the ASSM (TTRELDEF).

2. The optional word contained within the string is entered as the optional word associated with the relation (TTRELDEF).

3. The comment, if any, is associated with the relation-name in the ASSM (TTATTCOM).

MODIFication mode

1. The complement name, optional word, legal subject types, and legal object types are changed as specified in the following sentences.

2. The comment, if any, replaces the one in the ASSM (TTATTCOM).

3. If the comment is not given, any existing comment will be retained.

DELETion mode

1. The lists of legal subject and object types will be removed from the ASSM (TTRELDEF).

2. The complementary relation-name, its association with an optional word, if any, and its associated comment will be removed from the ASSM (TTRELDEF).

3. The relation-name, its association with an optional word, if any, and any associated comment will be removed from the ASSM (TTRELDEF).

Possible Semantic Errors

ADDition mode

1. The relation-name is already in use (449-TTRELDEF).

2. The optional word is not a legal optional word (449-TTGETOW).

3. No comment is specified (201-ACTION).

MODIFication mode

1. The name is not defined in the ASSM as a primary relation-name (408, 446-TTRELDEF).

2. The optional word is not a legal optional word (449-TTGETOW).

DELETion mode

1. The name is not defined in the ASSM as a primary relation-name (408, 446-TTRELDEF).

2. An instance of the relation exists between elements in the ASSM (400-TTRELDEF).

ALL modes

Extension permission has not been established (496-TTRELDEF).

<relation-definition sentence>::=

<complementary part>

| <subject part>

| <object part>

<complementary part>::=

COMPLEMENTARY $\left\{ \begin{matrix} \text{RELATION} \\ \text{RELATIONSHIP} \end{matrix} \right\}_1^1$ <relation id>.

Semantic Action

INSERT mode

1. The complementary relation-name is associated with the primary relation (TTRELCOMP).

2. The optional word is entered as the optional word associated with the complementary relation-name (TTRELCOMP).

REMOVE mode

The complementary relation-name, and its association with an optional word, if any, are deleted from the ASSM (TTRELCOMP).

Possible Semantic Errors

INSERT mode

1. The complementary relation-name is already in use (449-TTRELCOMP).

2. The optional word is not a legal optional word (449-TTGETOW).

REMOVE mode

The name is not defined in the ASSM as the complement of the primary relation-name (438-TTRELCOMP).

<subject part>::=

SUBJECT [ELEMENT_TYPE] [<element types>].

Semantic Action

INSERT mode

The list of element-types is entered into the ASSM as the set of legal subject types for the relation (TTRELSUBJ). (NOTE: See <attribute-definition sentence> for the semantics of <element types>.)

REMOVE mode

1. If no <element types> are specified, or ALL is specified, all subject element-types will be removed (TTRELSUBJ).

2. If a list is specified, those element-types on the list will be removed from the set of subject element-types (TTRELSUBJ).

Possible Semantic Errors

INSERT mode

1. No element-types are specified (463-TTRELSUBJ).

2. A comment is specified. The comment will be ignored, and the <subject part> processed (202-ACTION).

REMOVE mode

1. All EXCEPT is specified (452-TTRELSUBJ).

2. There is an instance of this relation with subject element of a type listed (470-TTRELDEF).

3. An element-type specified is not in the set of legal subject types (442-TTCHKETLM, TTCHKRSUBJ).

4. A comment is specified. The comment will be ignored, and the <subject part> processed (202-ACTION).

<object part>::=

OBJECT [ELEMENT_TYPE] [<element types>].

Semantic Action

INSERT mode

The list of element-types is entered into the ASSM as the set of legal object element-types for the relation. (NOTE: See <attribute-definition sentence> for the semantics of <element types>.)

REMOVE mode

1. If no element-types are specified, or ALL is specified, all object element-types will be removed (TTRELOBJ).

2. If a list is specified, those element-types on the list will be removed from the set of object element-types (TTRELOBJ).

Possible Semantic Errors

INSERT mode

1. No element-types are specified (463-TTRELOBJ).

2. A comment is specified. The comment will be ignored, and the <object part> processed (202-ACTION).

REMOVE mode

1. ALL EXCEPT is specified (452-TTRELOBJ).

2. There is an instance of this relation with object element of a type listed (470-TTRELOBJ).

3. An element-type specified is not in the set of legal object types (442-TTCHKETLM, TTCHKROBJ).

4. A comment is specified. The comment will be ignored, and the <object part> processed (202-ACTION).

<element definition>::=

<element-definition header>

$$\left\{ \begin{array}{l} [\langle\text{sentence keyword}\rangle] \\ \langle\text{element-definition sentence}\rangle \end{array} \right\}_{0}^{n}$$

Semantic Action

The header is processed according to the mode of the <section keyword> (explicit or default), and each sentence is processed according to the mode (explicit or default) of the <sentence keyword>.

Possible Semantic Errors

1. An <element-definition header> is followed by a sentence which is not an <element-definition sentence> (424-TTSNTCHK).

2. An <element-definition sentence> is not preceded by an <element-definition header> (425-TTSNTCHK).

3. An <element-definition sentence> appears in an <element definition> of mode DELETE (411-ACTION).

<element-definition header>::=

element-type-name element-name [comment].

Semantic Action

ADDition mode

1. The new element-name is entered into the ASSM with type corresponding to the element-type-name (TTELEMDEF).

2. The comment, if any, is associated with the element-name in the ASSM (TTATTCOM).

MODIFication mode

1. The comment, if any, replaces the one in the ASSM (TTATTCOM).

2. If the comment is not given, any existing comment will be retained.

DELETion mode

1. All attribute instances involving the element, and their associated comments, will be removed from the ASSM (TTELEMDEF).

2. The element-name and its associated comment will be deleted from the ASSM (TTELEMDEF).

Possible Semantic Errors

ADDition mode

The element-name is already in use (449-TTELEMDEF).

MODIFication mode

1. The element-name is not defined in the ASSM (445-ACTION, TTNULLERR).

2. The type of the element does not correspond to the element-type-name (443-TTCHKTYPE, TTCHKELTYPE).

DELETion mode

1. The element-name is not defined in the ASSM (445-ACTION, TTNULLERR).

2. The type of the element does not correspond to the element-type-name (443-TTCHKTYPE, TTCHKELTYPE).

3. The element is the subject or object of a relationship instance (465, 476-TTELEMDEF).

4. The element has an associated STRUCTURE or PATH, or the element is associated with a node or branch of a STRUCTURE or PATH (402-TTELEMDEF).

<element-definition sentence>::=

<attribute declaration>

| <relation declaration>

| <path declaration>

| <structure declaration>

<attribute declaration>::=

$$\text{attribute-name}\left[\left\{\begin{array}{l}\text{value-name}\\ \text{number}\\ \text{string}\end{array}\right\}_1^1 \quad [\text{comment}]\right].$$

Semantic Action

INSERT mode

1. An instance of the designated attribute is entered into the ASSM (TTATTRDCL).

2. The attribute value is attached to the attribute instance in the ASSM (TTATTRDCL).

3. The comment, if any, is associated with the attribute instance in the ASSM (TTATTCOM).

REMOVE mode

1. The comment, if any, associated with the attribute is removed from the ASSM (TTDELCOM).

2. The attribute and its value are removed from the ASSM (TTATTRDCL).

Possible Semantic Errors

INSERT mode

1. The type of the element is not on the list of applicable element-types for this attribute (401-TTATTRDCL).

2. The value is a value-name, and neither the value-name or NAMED appears on the list of attribute legal values (437-TTCHKAVAL).

3. The value is a number, and NUMERIC does not appear on the list of attribute legal values (437-TTCHKAVAL).

4. The value is a string, and TEXT does not appear on the list of attribute legal values (437-TTCHKAVAL).

5. An instance of this attribute already exists (413-TTATTRDCL).

REMOVE mode

1. An instance of this attribute does not appear in the ASSM for the applicable element (453-TTATTRDCL).

2. A value is specified (482-TTATTRDCL).

3. A comment is specified (202-ACTION).

<relation declaration>::=

relation-name <object list>.

<object list>::=

$\left\{ \text{[element-type-name] element-name [comment]} \right\}_{1}^{n}$

Semantic Action

INSERT mode

1. A relationship instance is entered in the ASSM for each element-name in the <object list> (TTRELDCL). (The subject is the element whose name is in the header.)

2. If the element-name was not previously defined, it is entered into the ASSM with the specified type (TTCHKELTYPE).

3. The comment is associated with the preceding relationship instance (TTATTCOM).

4. If the relation-name is a complementary relation, an instance of the primary relationship is entered, with subject and object reversed (TTRELDCL).

REMOVE mode

Each relationship instance which has an object on the <object list>, and subject element in the header, is removed from the ASSM, together with its associated comment (TTRELDCL, TTDELCOM).

Possible Semantic Errors

INSERT mode

1. The element-name in the header is not a legal subject type for the given relation (442-TTCHKETLM, TTCHKROBJ).

2. The element-name on the <object list> is not a legal subject type for the given relation (441-TTCHKETLM, TTCHKROBJ).

3. An instance of the relationship already exists with given subject and object (informative error only) (204-TTCHKROBJ).

4. The element is undefined and no type is specified (461-ACTION).

5. The type of an element does not match the preceding element-type-name (443-TTCHKELTYPE, TTCHKTYPE).

6. The relationship is EQUATES, and the given SYNONYM already EQUATES TO an element (477-TTRELDCL).

REMOVE mode

1. The given relationship instance does not exist (459-TTCHKROBJ).

2. The type of an element does not match the element-type-name (443-TTCHKELTYPE, TTCHKTYPE).

3. A comment is specified (202-ACTION).

<path declaration>::=

PATH $\left[\left\{\langle\text{validation node}\rangle\right\}_{1}^{n}\ \text{END}\right]$ [comment].

<validation node>::=

[element-type-name] element-name [comment]

Semantic Action

INSERT mode

1. A PATH structure is begun (TTPATHDCL).
2. A node of type NET-AND-PATH-FIRST is created, and associated with the element-name in the header (TTPATHKW).
3. A node is created for each <validation node>, and linked to the preceding NET-AND-PATH-FIRST node or <validation node>. The comment following the <validation node>, if any, is attached to it (TTNODELINK).
4. If there are no syntactic or semantic errors, the PATH structure is made permanent (TTPATHDCL).
5. The comment following the END, if any, is associated with the NET-AND-PATH-FIRST node (TTATTCOM).

REMOVE mode

The PATH structure associated with the element-name in the header is deleted from the ASSM (TTPATHDCL).

Possible Semantic Errors

INSERT mode

1. The element-name in the header is not of type VALIDATION_PATH (481-TTPATHKW).
2. The element already has an associated STRUCTURE or PATH (422-TTPATHKW).
3. The element-type-name in the <validation node> does not match the type of the following element-name (443-TTCHKELTYPE, TTCHKTYPE).
4. The element-type-name in the <validation node> is not legal for PATHs (480-TTVALNODE).
5. No <validation node> is specified (484-TTPATHDCL).

REMOVE mode

1. The element-name in the header does not have an associated VALIDATION_PATH (458-TTPATHKW).

2. A <validation node> is specified (483-TTPATHDCL).

3. A comment is specified (202-ACTION).

<structure declaration>::=

STRUCTURE $\left[ \left\{ \langle\text{node}\rangle \right\}_{2}^{n} \ \text{END} \right]$ [comment].

<node>::=

<element node>
| <terminator>
| <and node>
| <or node>
| <consider-or node>
| <for-each node>
| <select node>

Semantic Action

INSERT mode

1. A NET structure is begun (TTSTRDCL).

2. A node of type NET-AND-PATH-FIRST is created, and associated with the element-name in the header (TTSTRKW).

3. Each succeeding <node> is linked to the preceding one (TTNODELINK).

4. The comment following the END, if any, is associated with the NET-AND-PATH-FIRST node (TTATTCOM).

5. If there are no syntactic or semantic errors, the structure is made permanent (TTSTRDCL).

REMOVE mode

The R_NET or SUBNET structure associated with the element-name in the header is deleted from the ASSM (TTSTRDCL).

Possible Semantic Errors

INSERT mode

1. The element-name in the header is not of type R_NET or SUBNET (475-TTSTRKW).

2. The element already has an associated STRUCTURE or PATH (422-TTSTRKW).

3. There is no node other than an interface or <terminator> (473-TTSTRDCL).

4. The final node is not an <element node> of type OUTPUT_INTERFACE or a <terminator> (474-TTSTRDCL).

5. A <node> follows a <terminator> or OUTPUT_INTERFACE (450-TTSTRNSN, TTCBRANCH).

6. An <element node> of type INPUT_INTERFACE appears, and is not the first <node> following STRUCTURE (436-TTSTRSN, TTCBRANCH).

7. The element-name in the header is of type SUBNET and no RETURN node appears in the STRUCTURE (489-TTSTRDCL).

REMOVE mode

1. The element in the header does not have an associated R_NET or SUBNET (460-TTSTRKW).

2. A <node> is specified (456-TTSTRDCL).

3. A comment is specified (202-ACTION).

<element node>::=

[element-type-name] element-name [comment]

Semantic Action

INSERT mode

1. A node of the appropriate type is created and associated with the element-name (TTSTRKW).

2. The comment, if any, is associated with the node (TTATTCOM).

Possible Semantic Errors

INSERT mode

1. The element-type-name does not match the type of the element-name (443-TTCHKELTYPE).

2. The element-name is not of a NET legal type (427-TTELNODE).

REMOVE mode

This production is not allowed (456-TTSTRDCL).

<terminator>::=

TERMINATE [comment]

| RETURN [comment]

Semantic Action

INSERT mode

A node is created of type TERMINATE or RETURN and the comment, if any, is associated with the node (TTERMINODE).

Possible Semantic Errors

INSERT mode

1. A RETURN exists on an R_NET (469-TTERMNODE).

2. More than one RETURN exists on a SUBNET (490-TTERMNODE).

REMOVE mode

This production is not allowed (456-TTSTRDCL).

<and node>::=

DO [comment] <branch> $\left\{ \text{AND <branch>} \right\}_1^n$ END

Semantic Action

INSERT mode

1. A pair of AND-nodes is created in the ASSM (called the AND-head and the AND-tail). For a terminating node, only the AND-head is created (TTBEGAND).

2. The comment, if any, is associated with the AND-head (TTATTCOM).

3. Each <branch> is linked as the successor of the AND-head and the predecessor of the AND-tail, if present (TTNODELINK).

4. The AND-head will be linked to the predecessor of the <and node>, and the AND-tail, if present, will be linked to its successor (TTNODELINK).

Possible Semantic Errors

INSERT mode

Terminating and non-terminating branches are mixed (478-TTCONAND).

REMOVE mode

This production is not allowed (456-TTSTRDCL).

<branch>::=

$$\left\{ <\text{node}> \right\}_{1}^{n}$$

Semantic Action

INSERT mode

1. Each <node> is linked to its successor (TTNODELINK).

2. The first node on the <branch> will be linked to the predecessor of the <branch>, and the last node will be linked to the successor of the <branch> (TTNODELINK).

Possible Semantic Errors

INSERT mode

1. An INPUT_INTERFACE begins a <branch> (435-TTBBRANCH).

2. An INPUT_INTERFACE follows another <node> on a <branch> (436-TTCBRANCH, TTSTRSN).

3. A <node> follows a <terminator> or OUTPUT_INTERFACE (450-TTCBRANCH, TTSTRSN).

4. An INPUT_INTERFACE exists on a SUBNET (488-TTSTRSN,TTSTRFN).

REMOVE mode

This production is not allowed (456-TTSTRDCL).

<or node>::=

IF [comment] <conditional branch>
{OR <conditional branch>}$_0^n$
OTHERWISE [<branch>] END

## Semantic Action

### INSERT mode

1. A pair of OR-nodes is created in the ASSM (called the OR-head and the OR-tail). For a terminating node, only the OR-head is created (TTBEGOR).

2. The comment, if any, is associated with the OR-head (TTATTCOM).

3. Each <conditional branch> is linked as the successor of the OR-head and the predecessor of the OR-tail, if present (TTNODELINK).

4. The word OTHERWISE is associated with the following <branch> as if it were a <condition> (TTFINOR).

5. The OR-head will be linked to the predecessor of the <or node>, and the OR-tail, if present, will be linked to its successor (TTNODELINK).

## Possible Semantic Errors

### INSERT mode

Terminating and non-terminating branches are mixed (478-TTCONOR,TTFINOR).

### REMOVE mode

**This production is not allowed (456-TTSTRDCL).**

<conditional branch>::=

[integer] <condition> <branch>

Semantic Action

INSERT mode

1. The integer (ordinal), if any, is associated with the <branch> (TTBEGOR, TTCONOR).

2. The <condition> is associated with the <branch> (TTBEGOR, TTCONOR).

3. Data-names in the condition are associated with the <branch> (TTASSDATA).

Possible Semantic Errors

INSERT mode

The integer is greater than four digits (406-TTCONOR, TTBEGOR).

REMOVE mode

This production is not allowed (456-TTSTRDCL).

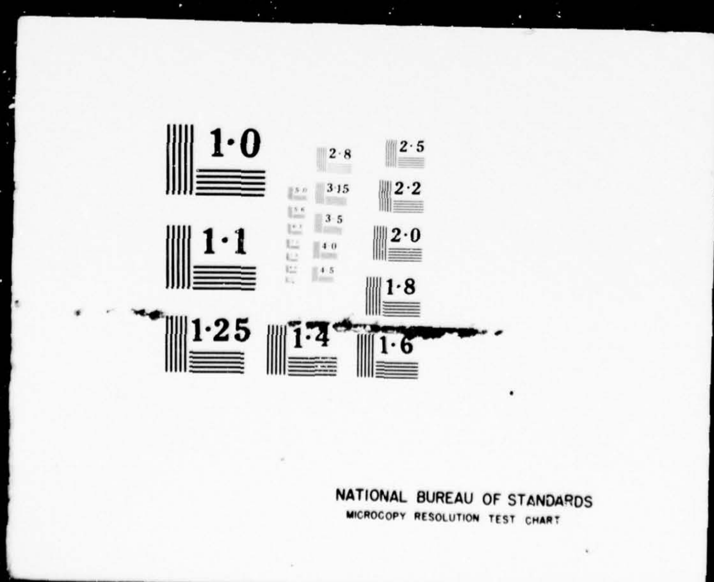
AD-A046 573 TRW DEFENSE AND SPACE SYSTEMS GROUP HUNTSVILLE ALA F/G 9/2
REVS MAINTENANCE MANUAL. SREP FINAL REPORT. VOLUME III.(U)
AUG 77 W E BENOIT, P N BERGSTRESSER DASG60-75-C-0022
UNCLASSIFIED TRW-27332-6921-026-VOL-3 NL

2 OF 5
ADA
046573

SIFIED

2 OF 5

ADA
046573



<consider-or node>::=

<consider-data>

<consider-data>::=

CONSIDER [DATA] enumerated-data-name IF [comment]

<consider-data branch> $\left\{ \text{OR <consider-data branch>} \right\}_{1}^{n}$

END

Semantic Action

INSERT mode

1. A pair of OR-nodes is created in the ASSM (called the OR-head and the OR-tail). For a terminating node, only the OR-head is created (TTBEGOR).

2. The comment, if any, is associated with the OR-head (TTBEGOR).

3. The enumerated-data-name is associated with the OR-head (ACTION).

4. Each <consider-data branch> is linked as the successor of the OR-head and the predecessor of the OR-tail, if present (TTNODELINK).

5. The OR-head will be linked to the predecessor of the <consider-or node>, and the OR-tail, if present, will be linked to its successor (TTNODELINK).

Possible Semantic Errors

INSERT mode

1. The enumerated-data-name is not an element of type DATA (492-ACTION, TTCONSIDER).

2. The enumerated-data-name has a value other than ENUMERATION for the attribute TYPE (493-TTCONSIDER).

3. Terminating and non-terminating branches are mixed (478-TTCONOR, TTFINOR).

4. The <consider-data> node has no non-empty branch (503-TTFINOR).

5. The <consider-data> node has more than one empty branch (504-TTCONOR).

REMOVE mode

This production is not allowed (456-TTSTRDCL).

<consider-data branch>::=

(<enumeration-value-list>) <branch>

| (<enumeration-value-list>)

<enumeration-value-list>::=

enumeration-value-name $\left\{ \text{OR enumeration-value-name} \right\}_{0}^{n}$

Semantic Action

INSERT mode

The <enumeration-value-list>, along with the surrounding parentheses, is entered as an alphanumeric string associated with the arc from the OR-head to the <branch>, if one is present, or associated with the arc from the OR-head to the OR-tail, if the <branch> is empty (TTBEGOR, TTCONOR).

Possible Semantic Errors

INSERT mode

1. A name specified is not an enumeration-value-name (495-TTASSEOREVNAME).

2. A name is duplicated in the <enumeration-value list>(505-TTEOREVNAME).

3. The <enumeration-value-list> contains illegal "blank" characters, i.e., commas, colons, or semicolons (506-TTCHKCOND).

REMOVE mode

This production is not allowed (456-TTSTRDCL).

<consider-or node>::=

<consider-entity-class>

<consider-entity-class>::=

CONSIDER [ENTITY_CLASS] entity-class-name IF [comment]

<consider-entity-class branch> $\left\{ \text{OR <consider-entity-class branch>} \right\}_{1}^{n}$

END

<u>Semantic Action</u>

<u>INSERT mode</u>

1. A pair of OR-nodes is created in the ASSM (called the OR-head and the OR-tail). For a terminating node, only the OR-head is created (TTBEGOR).

2. The comment, if any, is associated wtih the OR-head (TTBEGOR).

3. The entity-class-name is associated with the OR-head (ACTION).

4. Each <consider-entity-class branch> is linked as the successor of the OR-head and the predecessor of the OR-tail, if present (TTNODELINK).

5. The OR-head will be linked to the predecessor of the <consider-entity-class> node, and the OR-tail, if present, will be linked to its successor (TTNODELINK).

<u>Possible Semantic Errors</u>

<u>INSERT mode</u>

1. The entity-class-name is not an element of type ENTITY_CLASS (492-ACTION, TTCONSIDER).

2. Terminating and non-terminating branches are mixed (478-TTCONOR, TTFINOR).

3. The <consider-entity-class> node has no non-empty branch (503-TTFINOR).

4. The <consider-entity-class> node has more than one empty branch (504-TTCONOR).

<consider-entity-class branch>::=

(<entity-type-list>) <branch>

| (<entity-type-list>)

<entity-type-list>::=

entity-type-name $\left\{ \text{OR entity-type-name} \right\}_{0}^{n}$

<u>Semantic Action</u>

<u>INSERT mode</u>

The <entity-type-list>, along with the surrounding parentheses, is entered as an alphanumeric string associated with the arc from the OR-head to the <branch>, if one is present, or associated with the arc from the OR-head to the OR-tail, if the <branch> is empty (TTBEGOR, TTCONOR).

<u>Possible Semantic Errors</u>

<u>INSERT mode</u>

1. A name specified is of an element-type other than ENTITY_TYPE (494-TTASSEOREVNAME).

2. A name is duplicated in the <entity-type-list> (505-TTEOREVNAME).

3. The <entity-type-list> contains illegal "blank" characters, i.e., commas, colons, or semicolons (506-TTCHKCOND).

<for-each node>::=

FOR EACH <for-each subject> [SUCH THAT <condition>]

DO [comment] <for-each body node> END

Semantic Action

INSERT mode

1. A FOR-EACH-node is created in the ASSM (called the FOR-head) (TTFOREACH).
2. The comment, if any, is associated with the FOR-head (TTATTCOM).
3. The <condition> is associated with the branch from the FOR-head to the <for-each body node> (TTFOREACH).
4. Data-names in the <condition> are associated with the branch from the FOR-head to the <for-each body node> (TTASSDATA).

Possible Semantic Errors

REMOVE mode

This production is not allowed (456-TTSTRDCL).

<for-each subject>::=

    [FILE] file-name [RECORD]
  | [ENTITY_TYPE] entity-type-name
  | [ENTITY_CLASS] entity-class-name

<u>Semantic Action</u>

<u>INSERT mode</u>

An element of the appropriate type is created, if not previously defined, and associated with the FOR-head (TTCHKELTYPE, TTFOREACH).

<u>Possible Semantic Errors</u>

<u>INSERT mode</u>

1. The element-type-name does not match the type of the element-name (443-TTCHKELTYPE).

2. The element-type-name is not FILE, ENTITY_TYPE, or ENTITY_CLASS (433-TTFILENCHK).

3. The word RECORD is specified but the element-type-name is not FILE (431-TTFILENCHK).

<u>REMOVE mode</u>

This production is not allowed (456-TTSTRDCL).

<for-each body node>::=

[ALPHA] alpha-name [comment]

[SUBNET] subnet-name [comment]

Semantic Action

INSERT mode

1. A node of the appropriate type is created and associated with the ALPHA or SUBNET name (TTELNODE).

2. The comment, if any, is associated with the node (TTELNODE).

3. The node is attached as the successor of the FOR-head (TTNODELINK).

Possible Semantic Errors

1. The element-type-name does not match the type of the element-name (433-TTCHKELTYPE).

2. The element-type-name is not ALPHA or SUBNET (432-TTFEBODY).

REMOVE mode

This production is not allowed (456-TTSTRDCL).

<select node>::=

SELECT <select subject> SUCH THAT <condition> [comment]

Semantic Action

INSERT mode

1. A SELECT-node is created in the ASSM (TTSELECT).
2. The comment, if any, is associated with the SELECT-node (TTSELECT).
3. The <condition> is associated with the branch from the SELECT-node to its successor node (TTSELECT, TTNODELINK).

Possible Semantic Errors

REMOVE mode

This product is not allowed (456-TTSTROCL).

<select subject>::=

    [ENTITY_CLASS] entity-class-name
  | [ENTITY_TYPE] entity-type-name

Semantic Action

INSERT mode

An element of the appropriate type is created, if not previously defined, and associated with the SELECT-node (TTCHKELTYPE, TTSELECT).

Possible Semantic Errors

INSERT mode

1. The element-type-name does not match the type of the element-name (443-TTCHKELTYPE).

2. The element-type-name is not ENTITY_TYPE or ENTITY_CLASS (486-TTENTCHK).

REMOVE mode

This production is not allowed (456-TTSTRDCL).

```
<condition>::=
      (<Boolean expression>)
<Boolean expression>::=
      <simple Boolean> {<B op>  <simple Boolean>}^n_0
<simple Boolean>::=
      <Boolean term> {OR <Boolean term>}^n_0
<Boolean term>::=
      <Boolean factor> {AND <Boolean factor>}^n_0
<Boolean factor>::=
          <Boolean> [<rel op> <Boolean>]
      |   <arithmetic expression> <rel op> <arithmetic expression>
<Boolean>::=
      [NOT] <Boolean primary>
<Boolean primary>::=
          TRUE
      |   FALSE
      |   data-name
      |   (<Boolean expression>)
<arithmetic expression>::=
          [<ad op>] <arithmetic term>
      |   <arithmetic expression> {<ad op> <arithmetic term>}^n_0
<arithmetic term>::=
      <arithmetic factor> {<mul op> <arithmetic factor>}^n_0
<arithmetic factor>::=
          number
      |   data-name
      |   (<arithmetic expression>)
<B op>::=
      EQU | XOR
<rel op>::=
      = | < | > | < = | > = | <>
<ad op>::=
      + | -
<mul op>::=
      * | / | DIV | MOD
```

Semantic Action

INSERT mode

The <condition> is scanned, checked for syntactic validity, and stored as an alphanumeric string, including the surrounding parentheses (ACTION).

Possible Semantic Errors

INSERT mode

1. A data-name is not an element of type DATA (409-TTDATANAME).

2. The <condition> contains illegal "blank" characters, i.e., commas, colons, or semicolons (506-TTCHKCOND).

REMOVE mode

This production is not allowed (456-TTSTRDCL).

### 3.2.5 Error Handling

Errors are of three types: lexical, syntactic and semantic. A lexical error occurs during the processing of individual characters in the input stream by the lexical scanner, which is unable to combine them into the terminal symbols of the grammar (reserved-words, identifiers, punctuation, real numbers, integers, comments, and strings). A syntactic error occurs when legal terminal symbols are juxtaposed in a way which is not permitted by the grammar of RSL. A semantic error occurs when the meaning of a syntactically correct RSL command conflicts with that of a previous one.

In the case of a lexical or semantic error, the parser can issue a message and continue. However, recovery from a syntax error is complicated by the presence in the parse stack of partially parsed RSL commands. Error recovery is made by searching the input stream for a stop symbol, and then discarding all symbols in the parse stack above a corresponding continuing symbol. After this process is completed, an error recovery message is issued, and parsing continues. All text between a syntactic error and its recovery is ignored. In addition, some preceding text may have been effectively ignored because of the discarding of a portion of the parse stack. A list of stop and continuing symbols is given in Table 3.2.

Emptying of the parse stack must be done with care, since there may be information in the ASSM corresponding to a symbol in the parse stack. A CASE statement, indexed on the L-B CWS code for the symbol in the parse stack, is executed and information deleted from the ASSM in the appropriate cases.

The parse stack is initialized to contain <command-list head> and end-of-file as a stop symbol, so that recovery can always be made from a syntax error. The mechanism provided by the L-B CWS for cases when no error recovery is possible has been left in the RSL translator to aid in diagnosing any translator bugs, but is not expected to be executed.

Table 3.2 RSL Translation Stop and Continuing Symbols

| Stop Symbol | Continuing Symbol |
|---|---|
| . | <command> |
| . | <command-list head> |
| END | <and-node header> |
| OTHERWISE | <or-node header> |
| AND | <and-node header> |
| OR | <or-node header> |
| ) | ( |
| end-of-file | <command-list head> |

Error messages are controlled by the value of ERROR LEVEL as follows:

| | | |
|---|---|---|
| 0 | - | No error messages. |
| 1 | - | Error number and pointer. (The default value is 1.) |
| 2 | - | Dump of the parse stack when a syntax error is encountered, and again when recovery is made. |
| 3 | - | Dump of the parse stack whenever the lexical scanner is called. (This should only be used in extreme cases, as it produces a huge amount of output.) |

A pause stack dump will display the actual non-terminal and terminal symbols used as input to the Lecarme-Bochmann Compiler Writing System (L-B CWS) to define the syntax of the Requirements Statement Language (RSL). The complete syntax of RSL as input to the L-B CWS is contained on deck RSLDEF of the CWS Source Program Library (SPL). This SPL is maintained as file number fifteen on the REVS Software Deliverable File (see Sections 7.1 and 7.3.1).

## 3.3 INTERACTIVE R-NET GENERATION (RNETGEN)

### Description

Although REVS provides the user with the capability of defining an R-Net structure via a one-dimensional RSL input text string, the R-Net Generation function provides the REVS user with an alternate method of creating and maintaining R-Nets and Subnets. This method fulfills the need for creating/presenting a structure in a two-dimensional graphical representation. This is accomplished via an interactive graphics display system, namely, the Data Disc Color Graphics Display System [4]. This facility does not preclude, however, a structure declaration via the RSL structure syntax in the batch operating mode. It, in fact, supports the transformation of an RSL generated structure into its two-dimensional graphical representation via the Successor Node Module. Once a structure exists in this graphical form, it may be manipulated via the interactive terminal. That is, nodes may be added, deleted, moved about, disconnected, commented, etc. The entire net may be scrolled, displaying selected portions of the net as requested by the user. Modified structures may then replace the original structures in the ASSM if the user so desires.

### Input

| | | |
|---|---|---|
| MENU SELECTION | - | From a menu list, depicted in Figure 3-5, the user selects a menu entry via the trackball input facility, an input mechanism which allows the user to input an x,y screen position via the placement of a cross-hair cursor at any position on the screen. |
| ASSM | - | The user specified structure declaration for an R-Net/Subnet. |
| TRACKBALL/KEYBOARD ENTRY | - | Further inputs are required at the appropriate module levels and are documented in their corresponding sections. |

### Output

| | | |
|---|---|---|
| NET/NODE DISPLAY | - | Net/nodes are displayed on the screen as requested by the user via the module level. |

ASSM - The R-Net/Subnet structure declaration is saved or updated according to user inputs at the module level.

Processing

Figure 3-6 is a flow diagram for the R-Net Generation function. The following comments further clarify the processing for the indicated steps.

[1] - Coordinates use of the graphics screen area with the REVS Executive.

[2] - If REVS is in the off-line mode, an error condition exists since RNETGEN can be executed only in the on-line mode.

[3] - All flags used by RNETGEN are initialized. The default color selection is set to turquoise. All ASSM pointers required by RNETGEN are retrieved for subsequent access. The menu is displayed on the CRT and color codes for zoomed-out displays are appropriately initialized.

Procedure References

The following correlates the functional processing steps shown in Figure 3-6, with the REVS procedures which perform the indicated processing.

[1] - XXREVSGRAPH

[3] - IINITIAL

[4] - IIMENUREAD, IINSGOUT, IICHKTYPNOD, IIRESETM

[6] - XXREVSOUT

[7] - IISTRTYPE

[8] - IICRNODE

[9] - IIMVNODE

[10] - IIJNNODE

[11] - IIDJNODE

[12] - IISAVE

[13] - IICOLOR

| | | |
|---|---|---|
| [14] | - | IIDENODE |
| [15] | - | IICMNODE |
| [16] | - | IISUNODE |
| [17] | - | IISCROLL |
| [18] | - | IIZOOMIN |
| [19] | - | IIZOOMOUT |
| [20] | - | IIDSNODE |
| [21] | - | IICALCOMP |
| [22] | - | IIDSPBRN |
| [23] | - | IISTOP |
| [24] | - | IIAUTOPLOT |
| [25] | - | IIMVSUBTREE |

MENU

- STRUCTURE TYPES
  - RNET
  - SUBNET
  - VALPATH
- NODE TYPES
  - INPUT
  - VALPT
  - ALPHA
  - EVENT
  - SUBNET
  - OUTPUT
  - SELECT
  - OR
  - FOR
  - AND
  - FIRST
  - TERMINAL
  - RETURN
  - OTHER
- MOVE SUBTREE
- MOVE NODE
- CONNECT NODES
- DISCONNECT NODES
- DELETE NODE
- COMMENT NODE
- SUCCESSOR NODE
- DISPLAY NODE
- DISPLAY BRANCH
- SCROLL NET
- AUTOPLOT
- ZOOM-IN ON NET
- ZOOM-OUT ON NET
- CALCOMP
- SAVE NET
- STOP

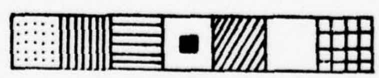


Figure 3-5 RNETGEN Menu

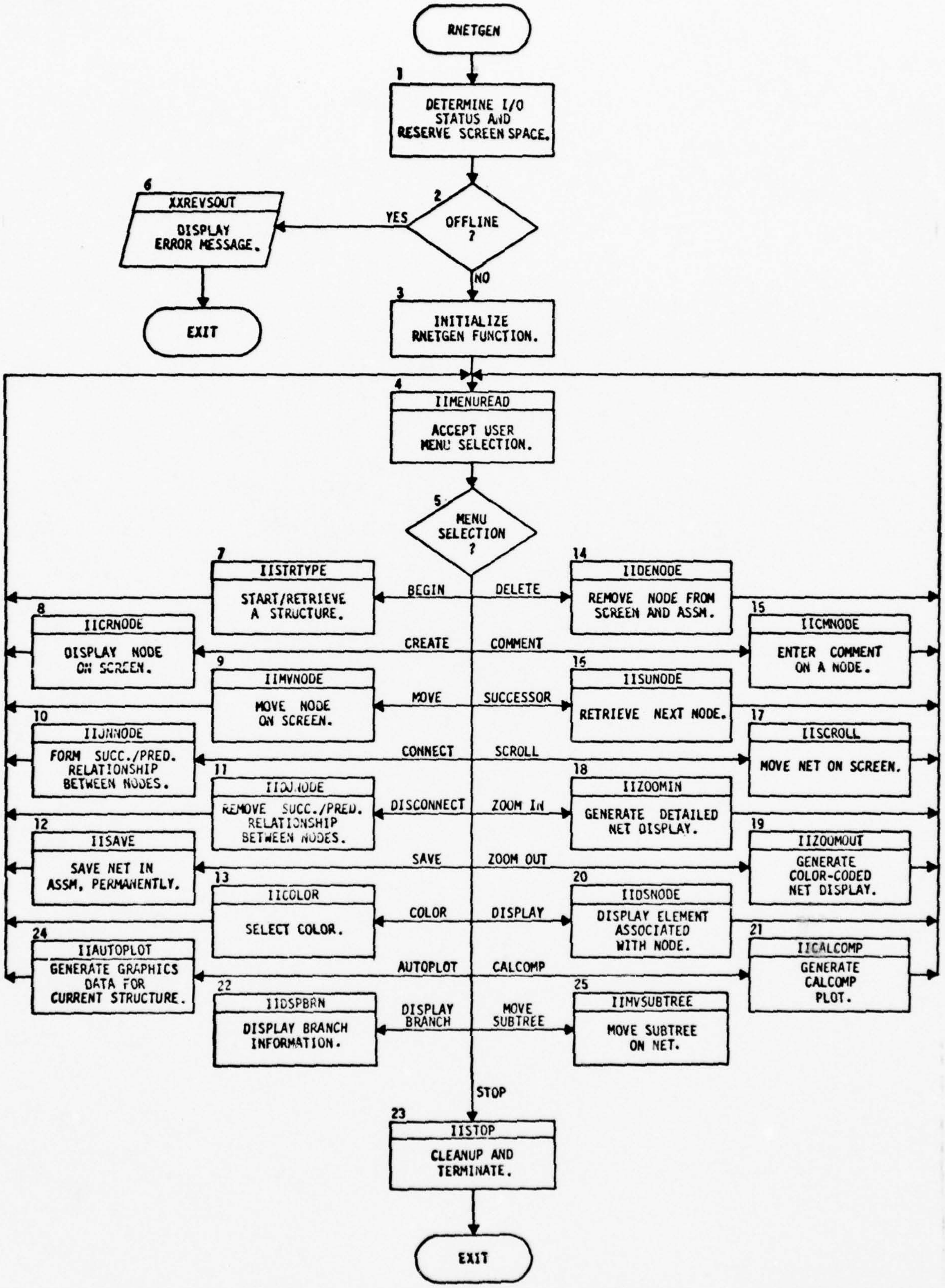



Figure 3-6 R-Net Generation (RNETGEN)

### 3.3.1 Begin Structure (IISTRTYPE)

Description

Upon input of the structure type and associated element name, this module will determine the existence of such a structure in the ASSM. If it does exist, it will be retrieved and displayed on the CRT, otherwise an entry node will be created for it and displayed at the top center of the CRT. The user may then add to or alter the structure via other menu selections.

Input

| | | |
|---|---|---|
| STRUCTURE TYPE | - | Desired structure type as selected from the available structure types in the menu. |
| ELEMENT NAME | - | Element name of ASSM element owning the structure. |

Output

| | | |
|---|---|---|
| ASSM | - | Temporary copy of structure if one already existed in ASSM. |
| NET DISPLAY | - | Display of the structure on the CRT. |
| SCREEN MATRIX | - | As nodes are displayed on the CRT, their corresponding ASSM pointers are entered into the corresponding screen matrix element. |
| COORDINATE TRANSFORMATION PARAMETERS | - | x,y transformation parameters required to translate from the current screen position to the initial screen position are initialized to zero. |

Processing

The flow diagram for this module is presented in Figure 3-7. Following is a further description of selected processing steps in the flow diagram.

| | | |
|---|---|---|
| [6,7] | - | A message, requesting the user to key-in via the keyboard the element name owning the desired structure, is displayed. |
| [8] | - | RSL element names must begin with an alphabetic character and can only contain alphanumeric characters with the exception of the underscore. |

[9-18] - If the element name which was keyed in is not found in the ASSM, the user is given the option to allow RNETGEN to enter it in the ASSM or to completely ignore the menu operation, in which case the module returns control back to the function level.

[19, 20, 24, 25] - If the element name is found to already exist in the ASSM, then the user is given the option to allow RNETGEN to retrieve its associated structure or to completely ignore the menu operation, in which case the module returns control back to the function level.

[26, 27, 33, 34, 36] - If the selected element has an associated structure, it is copied to the ASSM temporary structure area, otherwise, the first node is created in the ASSM and displayed at the top center of the screen display area.

[28, 29, 30, 31, 35] - If the selected structure has no graphics data associated with it, the user is given the option to allow the graphics data to be automatically generated via IIAUTOPLOT or to use the prompting capability via SUCCESSOR node menu entry.

[32, 37, 38] - If the structure has graphics data associated with it, the user is given the option to display the structure in either its zoomed-in or its zoomed-out mode.

Procedure References

The following correlates the functional processing steps shown in Figure 3-7 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [4] | - | IICLEARSCREEN |
| [5] | - | IICLEARMATRIX |
| [6] | - | IIMSGOUT |
| [7] | - | IIMSGIN |
| [8] | - | IICHKNAME |
| [10, 12, 13] | - | IIMSGOUT |
| [14] | - | IIMSGIN |
| [18] | - | IIDISFRST |
| [20, 21, 22, 23] | - | IIMSGOUT |
| [24] | - | IIMSGEN |
| [29] | - | IIMSGOUT |
| [31] | - | XXCNET |
| [33, 35] | - | IIMSGOUT |
| [36] | - | IIDISFRST |
| [37] | - | IIZOOMOUT |
| [38] | - | IIDISNET |

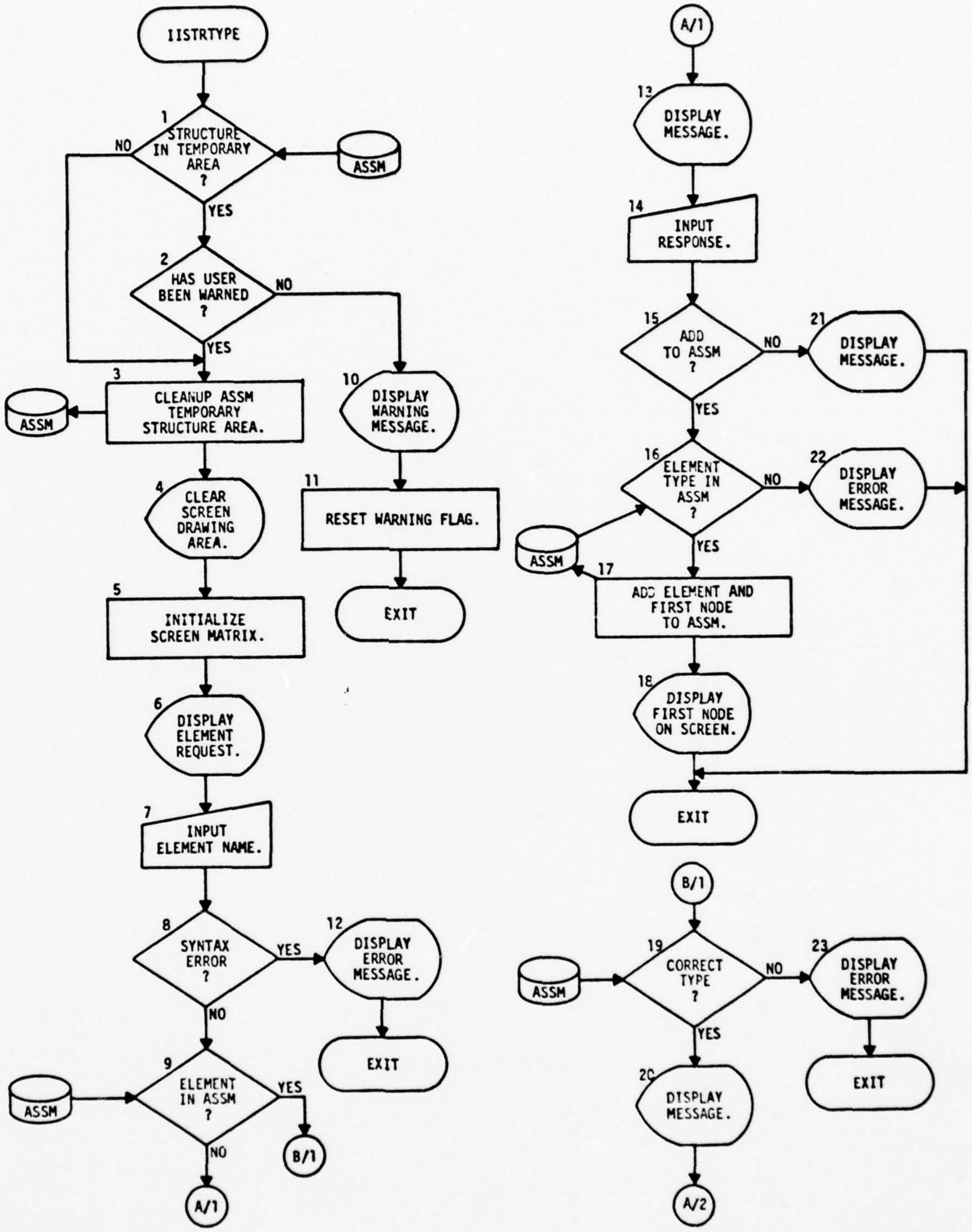



Figure 3-7 Begin Structure (IISTRTYPE)

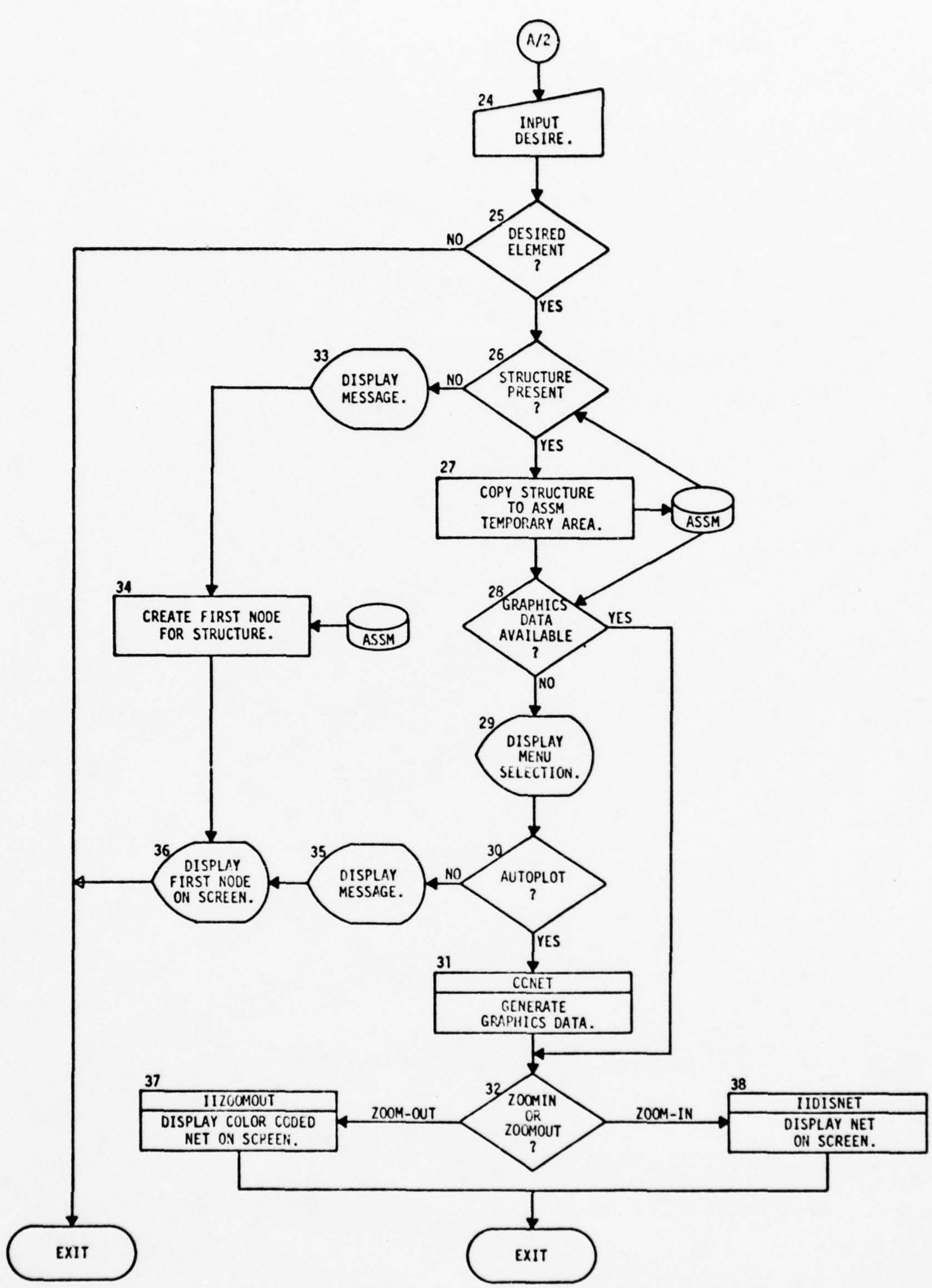



Figure 3-7 Begin Structure (IISTRTYPE) (Continued)

### 3.3.2 Create Node (IICRNODE)

Description

Upon input of the node type, associated element name, color, and screen position provided by the user, this module will display the node at the indicated x,y screen position. Input checks are performed to insure legality of the selected x,y position. An ASSM node record is created and the internal screen matrix is updated accordingly.

Input

| | | |
|---|---|---|
| NODE COLOR | - | The user may specify node color for display from a displayed list of available colors. The selection is made at the function level. |
| NODE POSITION | - | This input is provided by the user via a trackball entry indicating the desired x,y position for the node on the screen. |
| SCREEN MATRIX | - | Screen positional matrix and ASSM correlation matrix. |
| NODE DESCRIPTION | - | Node physical characteristics. |
| SCREEN LIMITS | - | Defines screen drawing area. |
| NODE TYPE | - | Desired node type as selected from the menu. |
| ELEMENT NAME | - | Associated ASSM element, if applicable. |

Output

| | | |
|---|---|---|
| ASSM | - | Node record containing node type, color, and x,y position. |
| NODE DISPLAY | - | Visual representation of the node at the selected x,y position on the screen. |
| SCREEN MATRIX | - | Node record pointer is entered into the appropriate screen matrix element. |

Processing

The flow diagram for this module is given in Figure 3-8. Following is a description of selected processing steps in the flow diagram.

[2] - Checks are performed to insure that the node will fit on the screen at the selected position and that it will not overlap existing nodes on the structure.

[3] - A check is performed to insure that the selected node is legal for the current structure type.

[4, 20] - If the node is an 'OR' node, the user is given the option to associate it to a data element.

[5, 21] - If the node is a 'FOR EACH' node, the user must specify its associated element type.

[7] - The associated element name is keyed in via the keyboard.

[8] - The keyed in element name must begin with an alphabetic character and can only contain alphanumeric characters with exception of the underscore, otherwise the menu operation is ignored.

[10-13] - If the specified element name is not to be found in the ASSM, the user is given the option to allow RNETGEN to enter it or to completely ignore the menu operation.

[22-25] - If the specified element name is already in the ASSM, the user is given the option to allow RNETGEN to associate the node with it or to completely ignore the menu operation.

Procedure References

The following correlates the functional processing steps shown in Figure 3-8 with the REVS procedures which perform the indicated processing.

[1] - IICMPMAT

[2] - IIMATCHK, IICHKNODESEL

[3] - IICHKNODTYP

[7] - IIMSGIN

[8] - IICHKNAME

| | | |
|---|---|---|
| [10] | - | IIMSGOUT |
| [11] | - | IIMSGIN |
| [16] | - | IINODIS |
| [19] | - | IIDISNAM |
| [23] | - | IIMSGOUT |
| [24] | - | IIMSGIN |
| [26] | - | IIMSGOUT |

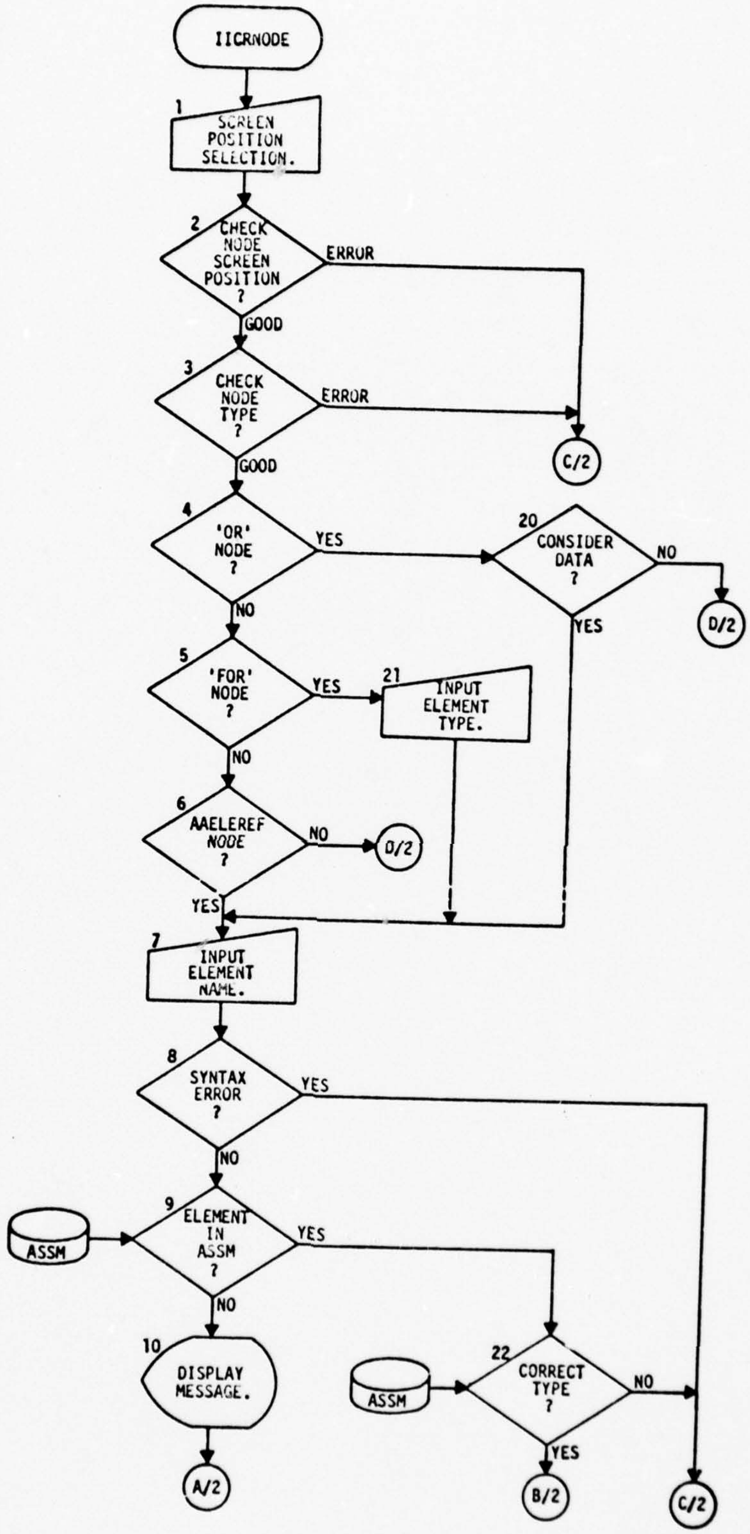



Figure 3-8 Create Node (IICRNODE)

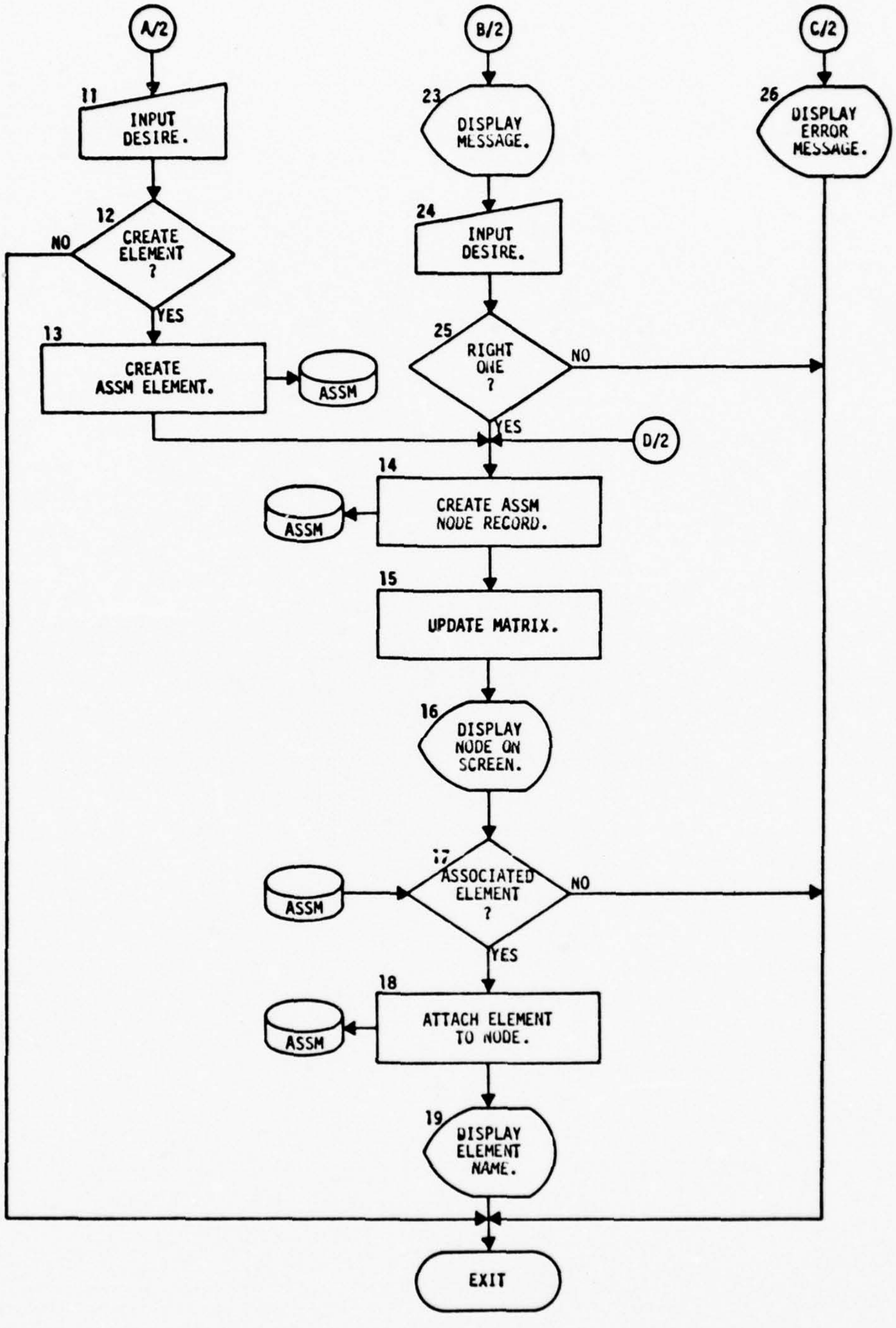



Figure 3-8 Create Node (IICRNODE) (Continued)

### 3.3.3 Delete Node (IIDENODE)

Description

The indicated screen selection is verified after which the selected node is removed from the screen, together with all its associated arcs. The node is subsequently removed from the ASSM structure and the internal screen matrix is updated accordingly.

Input

| | | |
|---|---|---|
| NODE SELECTION | - | The user specifies the node to be deleted from the net via the trackball input facility. |
| ASSM | - | Node record data and associated successor/predecessor record data for the selected node. |
| SCREEN MATRIX | - | Node positional and ASSM correlation data. |
| SCREEN LIMITS | - | Defines screen drawing area. |
| NODE DESCRIPTIONS | - | Node physical characteristics. |

Outputs

| | | |
|---|---|---|
| ASSM | - | Corresponding ASSM node record is removed from the ASSM structure. Also, any associated successor/predecessor records are removed. |
| SCREEN MATRIX | - | Updated accordingly. |
| DISPLAY | - | The selected node is removed from the screen. |

Processing

A flow diagram of this module is presented in Figure 3-9. The following comments apply to the indicated boxes of the flow diagram.

| | | |
|---|---|---|
| [2] | - | An input check is performed to insure an existing node was selected. |
| [3] | - | The selected node is removed from the screen along with all its associated arcs. |

[4] - The ASSM structure is updated to reflect the node deletion. Any successor/predecessor relationships with this node are removed. The node/ASSM element relationship is also removed, if one exists. Finally, the node record, itself, is removed from the ASSM.

Procedure References

The following correlates the functional processing steps shown in Figure 3-9 with the REVS procedures which perform the indicated processing.

[1] - IICMPMAT

[2] - IIMATCHK

[3] - IISETNODE, IINODIS

[6] - IIMSGOUT

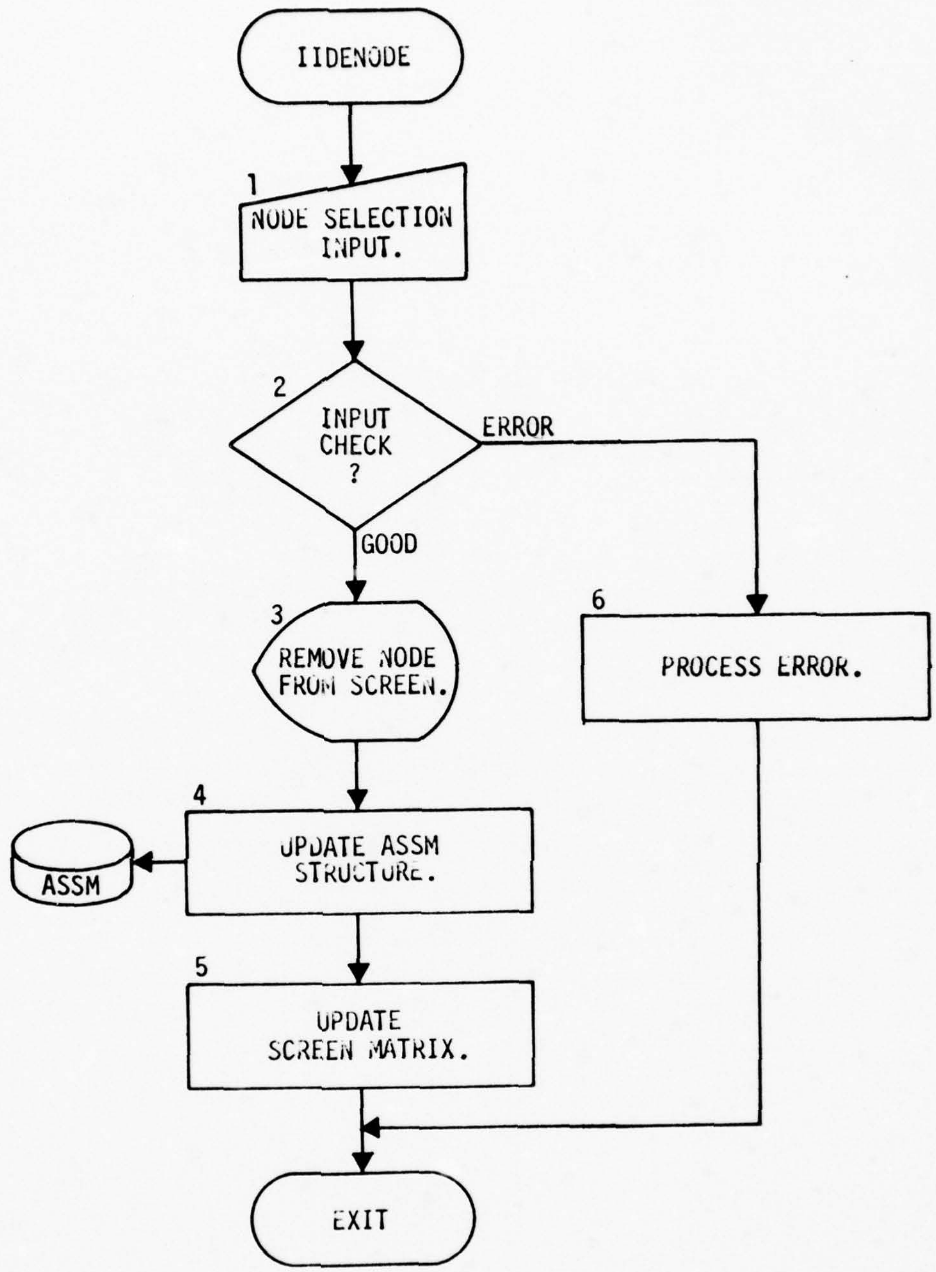



Figure 3-9 Delete Node (IIDENODE)

### 3.3.4 Move Node (IIMVNODE)

Description

Upon verification of a selected node and the 'to' x,y screen position, the indicated node and all its associated arcs are removed from its current position on the screen and subsequently redrawn at the desired 'to' screen position. The ASSM node and connector records are updated to reflect the new screen position of the node.

Input

| | | |
|---|---|---|
| NODE SELECTION | - | The user specifies the node to be moved via an x,y screen position using the trackball entry key. |
| ASSM | - | Node record data and associated successor/predecessor record(s). |
| SCREEN MATRIX | - | Node positional and ASSM correlation data. |
| SCREEN LIMITS | - | Defines screen drawing area. |
| NODE DESCRIPTIONS | - | Node physical characteristics. |
| SCREEN SELECTION | - | The user specifies the x,y 'to' position on the screen to which the node will be moved. |

Output

| | | |
|---|---|---|
| ASSM | - | Node record data and associated successor/predecessor record(s) data. |
| NODE DISPLAY | - | Removal of selected node at current screen position and display of node at newly selected screen position. |
| SCREEN MATRIX | - | Updated screen matrix. |

Processing

Figure 3-10 is a flow diagram for this module. The following comments further clarify the processing for the indicated steps.

| | | |
|---|---|---|
| [1] | - | The user inputs the desired node on the screen to be moved via an x,y screen position selection using the trackball facility. |

[2] - A validity check on input is performed. First, to determine whether the selected x,y position is within the screen drawing limits; and, secondly, to determine if a node exists at the selected x,y position.

[3] - The user inputs the desired x,y screen position to which the node selected in Step [1] is to be moved. This again is accomplished via the trackball facility.

[4] - An input check is performed to insure that the selected x,y position is within the screen limits. A check is also made to insure against node overlap of existing nodes on the screen.

[7] - Internal screen matrix is updated to reflect the new x,y position of the node.

[8] - The ASSM node record and associated successor/predecessor records are updated accordingly.

[9] - The node is redrawn on the screen at its new x,y position and control is returned to the function level.

<u>Procedure References</u>

The following correlates the functional processing steps shown in Figure 3-10 with the REVS procedures which perform the indicated processing.

[1] - IICMPMAT

[2] - IIMATCHK

[3] - IICMPMAT

[4] - IIMATCHK, IICHKNODESEL

[5] - IISETNODE

[6] - IINODIS, IISCNCHK, IILINE

[7, 8] - IICMPINT

[9] - IINODIS, IIDISNAM, IISCNCHK, IILINE

[10] - IIMSGOUT

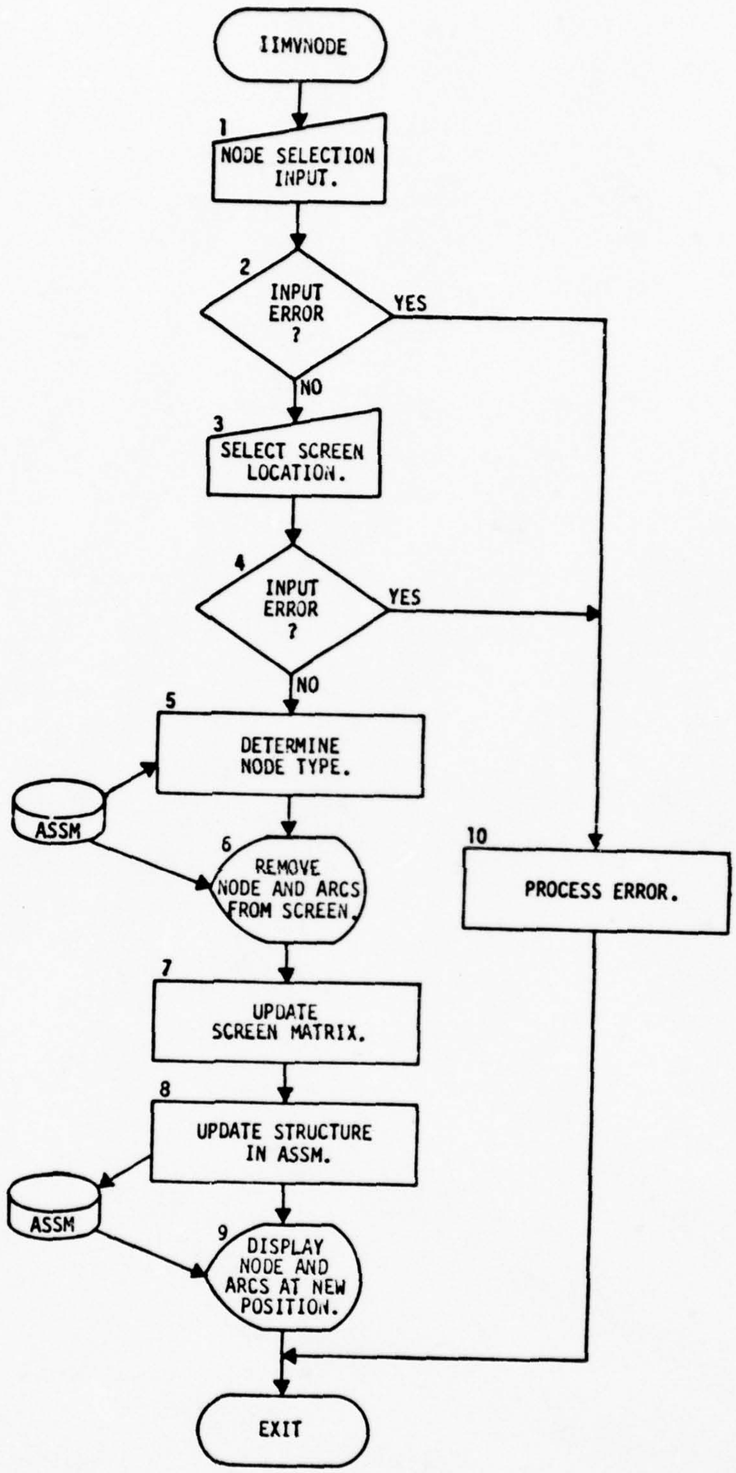



Figure 3-10 Move Node (IIMVNODE)

### 3.3.5 Join Nodes (IIJNNODE)

Description

Upon identification of the 'from' and 'to' nodes on the screen, checks are performed to determine the legality of the selections. If errors are detected, the input selection sequence must be repeated. Node intersection points are computed and a node connector record containing these intersection points is created and entered in the ASSM. The directed arc is subsequently displayed on the screen. In the case of an out-branching 'OR/FOR' node, the user must specify a branching condition which will also be entered in the ASSM.

Input

| | | |
|---|---|---|
| NODE SELECTIONS | - | The user specifies the nodes to be connected via the trackball facility. The node selection sequence also implies the direction for the directed arc. |
| ASSM | - | Node record data for the selected nodes. |
| CONDITIONAL | - | In the event of an 'OR/FOR' node, the user must provide the conditional branch data via the Anagraph keyboard. |
| SCREEN MATRIX | - | Node positional and ASSM correlation data. |
| NODE DESCRIPTIONS | - | Node physical characteristics. |
| SCREEN LIMITS | - | Defines screen drawing area. |

Output

| | | |
|---|---|---|
| ASSM | - | A node connector record is entered into the ASSM, together with appropriate branching condition, if the node is a branch node of an 'OR/FOR' node. |
| DISPLAY | - | The directed arc between the two nodes is displayed on the screen. |

Processing

Figure 3-11 presents a flow diagram of the processing for this module. Following are comments which apply to selected boxes in the flow diagram.

[2] - Checks are made to insure that a node does exist at the selected screen position and that it may legally be used as a predecessor node.

[4] - Checks are made to insure that a node does indeed exist at the selected screen position and that it may legally be used as a successor to the previously selected predecessor node.

[7-15] - If the predecessor node is an 'OR/FOR EACH' node, then the user enters the associated conditional expression and ordinal, if applicable.

[9] - A syntax check is performed on the conditional expression.

Procedure References

The following correlates the functional processing steps shown in Figure 3-11 with the REVS procedures which perform the indicated processing.

[1] - IICMPMAT

[2] - IIMATCHK, IICHKPRED

[3] - IICMPMAT

[4] - IICHKNODESEL, IIMATCHK, IICHKSUC

[8] - IICONDIN

[9] - IICHKSYNTAX

[13] - IIORDIN

[16] - IICMPINT

[17] - IILINE

[18] - IIMSGOUT

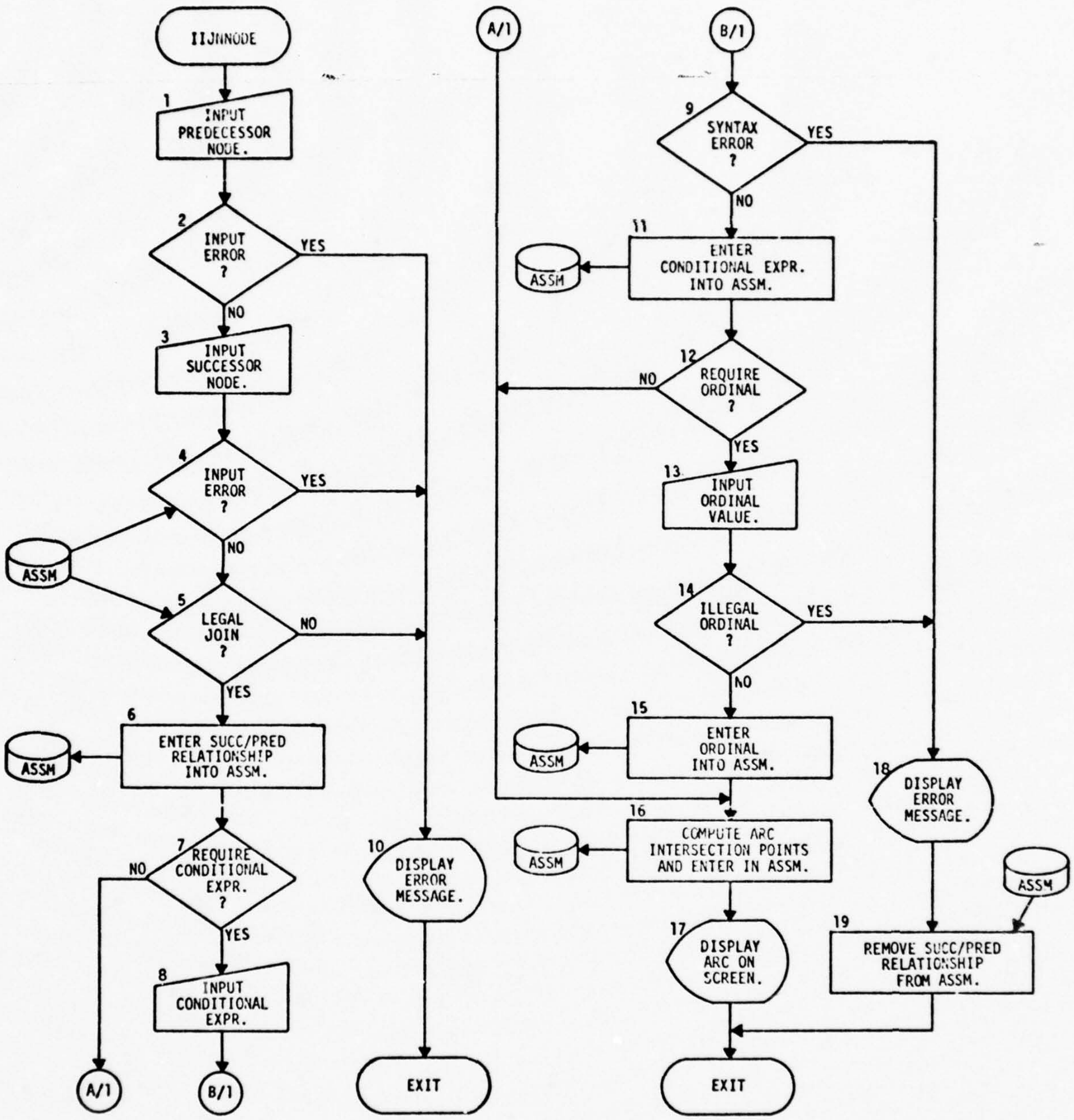



Figure 3-11 Join Nodes (IIJNNODE)

### 3.3.6 Disjoin Nodes (IIDJNODE)

#### Description

Upon verification of the node selections, this module removes successor/predecessor relationships between the indicated nodes, both in the ASSM structure and visually on the screen.

#### Input

| | | |
|---|---|---|
| NODE SELECTION | - | The user selects the two nodes to be disconnected via the trackball input facility. |
| SCREEN MATRIX | - | Node positional and ASSM correlation data. |
| SCREEN LIMITS | - | Defines screen drawing area. |

#### Outputs

| | | |
|---|---|---|
| ASSM | - | The successor/predecessor record data is removed from the ASSM structure for the indicated nodes. |
| DISPLAY | - | The directed arc between the two nodes is removed from the screen. |

#### Processing

Figure 3-12 is a flow diagram of the processing for this module.

#### Procedure References

The following correlates the functional processing steps shown in Figure 3-12 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | IICMPMAT |
| [2] | - | IIMATCHK |
| [3] | - | IICMPMAT |
| [4] | - | IIMATCHK, IICHKNODESEL |
| [5] | - | IILINE |
| [7] | - | IIMSGOUT |

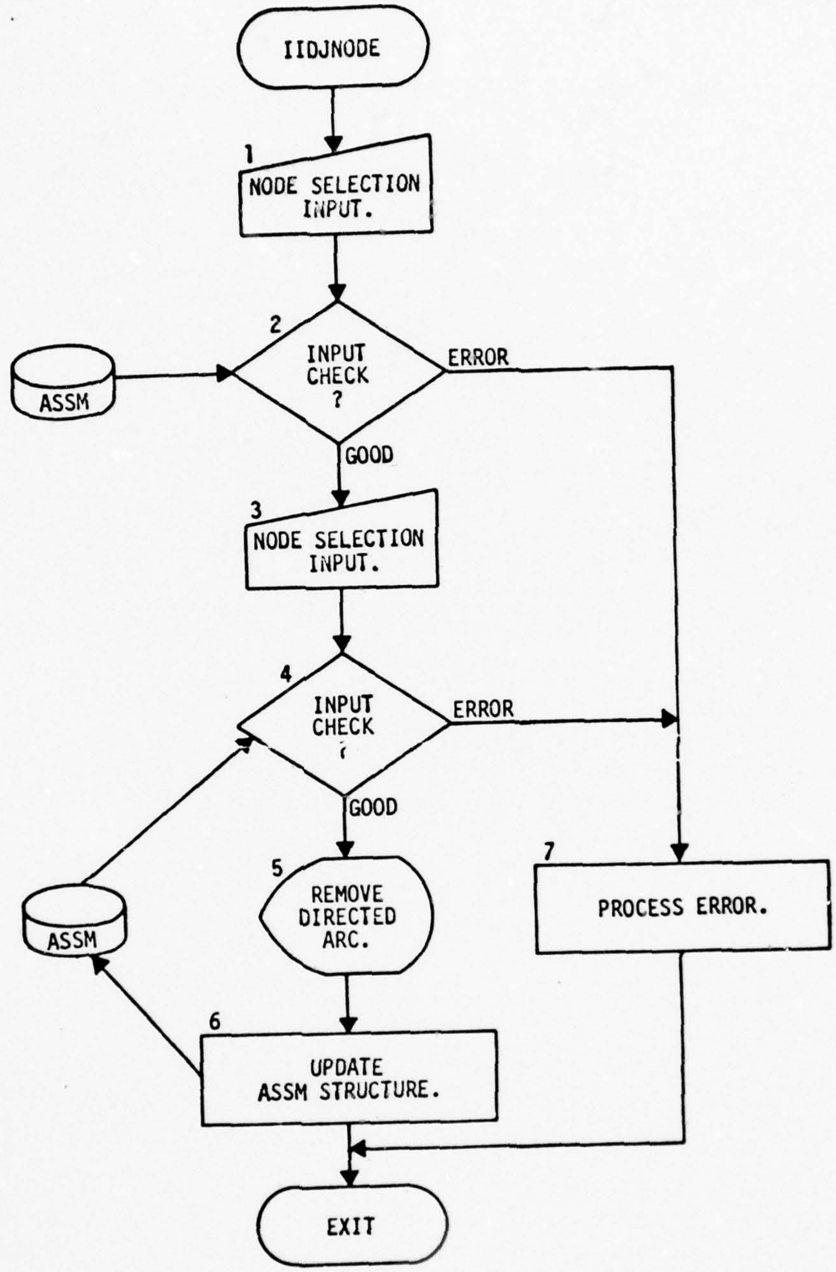



Figure 3-12 Disjoin Nodes (IIDJNODE)

3.3.7 Comment Node (IICMNODE)

Description

This module provides a mechanism for allowing the user to insert, display, or remove a comment with any node on a net structure. This is accomplished by selecting the applicable node via the trackball input facility and subsequently selecting the desired operation via the trackball.

Input

| | | |
|---|---|---|
| NODE SELECTION | - | The user selects the applicable node on the screen via the trackball. |
| COMMENT | - | Textual data is input by the user via the Anagraph keyboard. |
| SCREEN MATRIX | - | Node positional and ASSM correlation data. |
| SCREEN LIMITS | - | Defines screen drawing area. |
| OPERATION SELECTION | - | The user selects the desired operation using the trackball input facility. |

Output

| | | |
|---|---|---|
| ASSM | - | Comment is added/removed/displayed from the applicable node in the ASSM structure. |

Processing

The processing performed by this module is shown in Figure 3-13. In Step 2, an input check is performed to determine whether the selected x,y screen position is within the screen drawing area. A further check is made to insure that a node does, indeed, exist at the selected x,y position.

Procedure References

The following correlates the functional processing steps shown in Figure 3-13 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | IICMPMAT |
| [2] | - | IIMATCHK |
| [7] | - | IICOMENTIN |
| [11, 13, 14, 15] | - | IIMSGOUT |

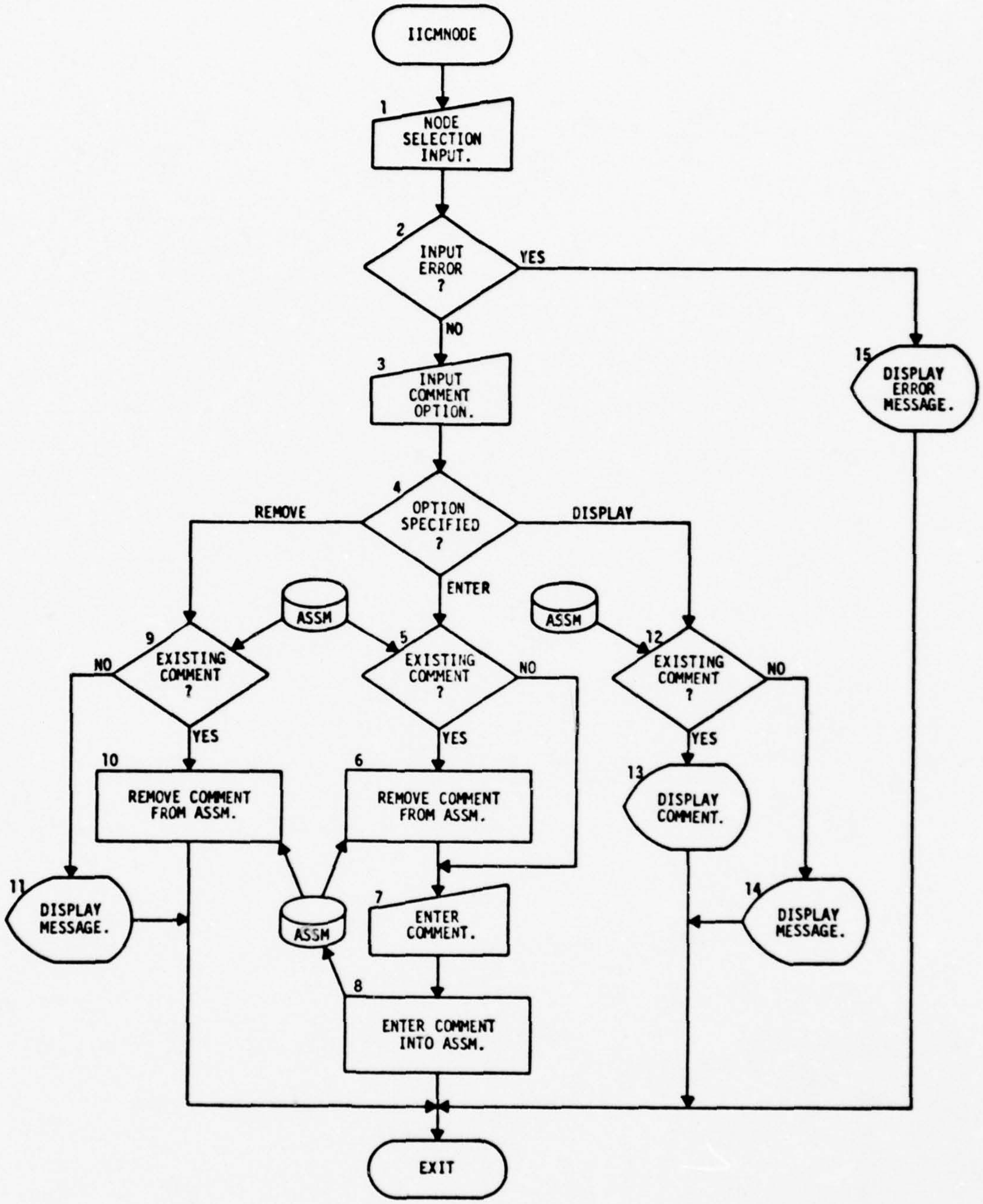



Figure 3-13 Comment Node (IICMNODE)

### 3.3.8 Successor Node (IISUNODE)

Description

This module is used to display on the screen a structure which was created in the batch (off-line) mode via the RSL translator. Such nodes have no graphics coordinate data or color associated with them. After having requested such a structure via the menu, the entry node will be displayed at the top center portion of the structure display area. The user should then select the Successor Node Module, via the menu, followed by successive trackball selections of the nodes for which successors are not displayed. The type of the successor node is identified and its desired position on the screen is then selected by the user.

Input

| | | |
|---|---|---|
| NODE SELECTION | - | The user selects the applicable node for which he wishes to display its successor. |
| ASSM | - | The entire ASSM net structure. |
| SCREEN POSITION | - | The user selects an x,y screen position at which the successor node is to be displayed (trackball input). |
| SCREEN MATRIX | - | Node positional and ASSM correlation data. |
| SCREEN LIMITS | - | Defines screen drawing area. |
| NODE DESCRIPTION | - | Node physical characteristics. |

Outputs

| | | |
|---|---|---|
| ASSM | - | x,y screen positions are added to the ASSM. |
| DISPLAY | - | Node is displayed on the screen. |
| SCREEN MATRIX | - | Screen matrix is updated accordingly. |

Processing

The processing performed by this module is shown in Figure 3-14. The following comments clarify the indicated processing step.

[2] - An input check is performed to determine whether the x,y input coordinate is within the screen drawing area and also to insure that a node does, indeed, exist at the selected x,y position.

[4] - A check is made to determine if more successors exist at the selected node.

[5] - The user is informed of the node type for the next successor.

[6] - An x,y screen position at which the node is to be displayed is selected by the user.

[7] - A check is performed to insure that the selected x,y position is within the screen drawing area and that the entire node will fit within the screen limits. A check is also made to insure against node overlap with existing nodes on the screen.

Procedure References

The following correlates the functional processing steps shown in Figure 3-14 with the REVS procedures which perform the indicated processing.

[1] - IICMPMAT

[2] - IIMATCHK

[6] - IICMPMAT

[7] - IICHKNODESEL, IIMATCHK

[10] - IINODIS, IIDISNAM

[11] - IICMPINT, IISCNCHK, IILINE

[12, 13] - IIMSGOUT

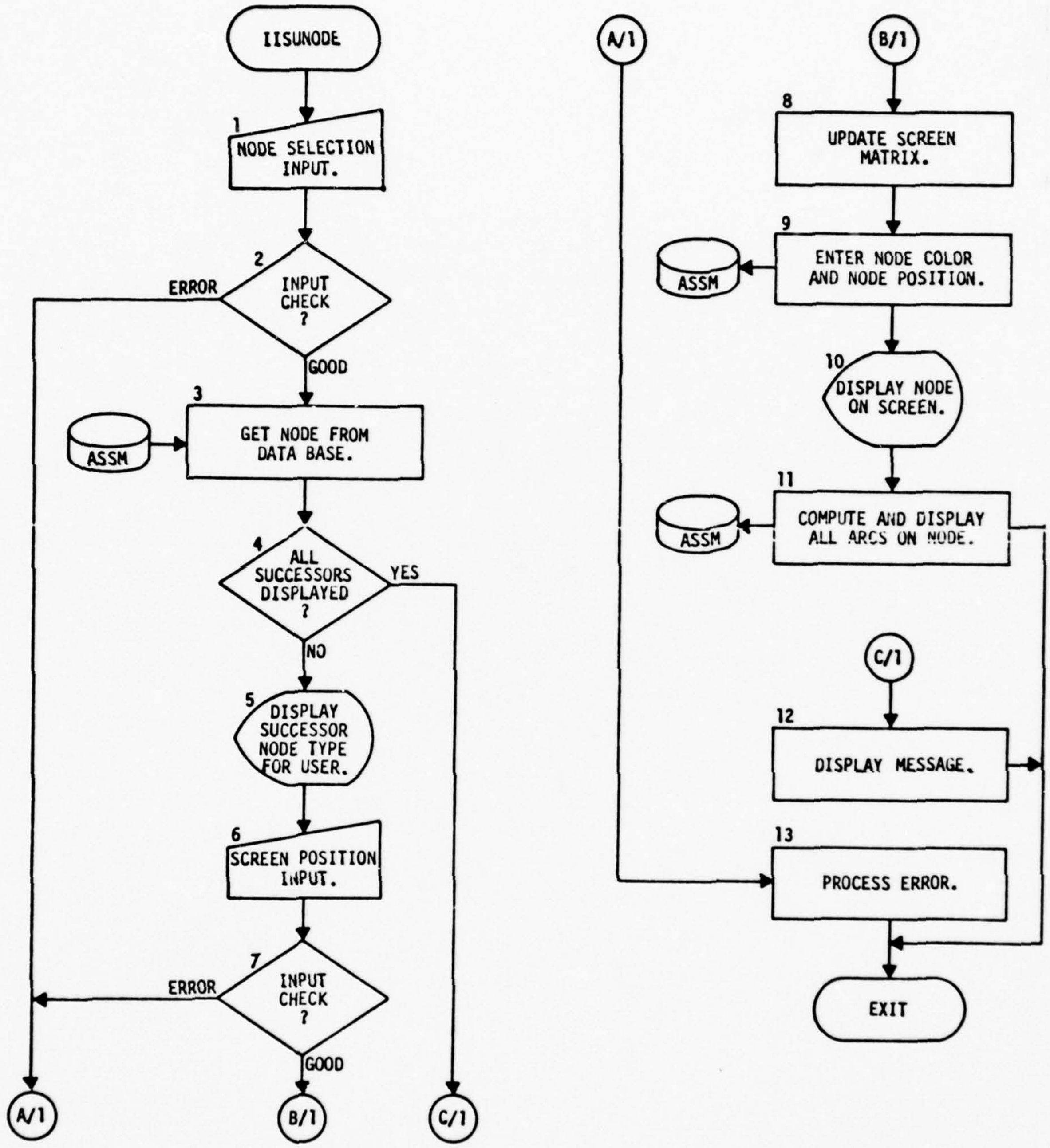



Figure 3-14 Successor Node (IISUNODE)

### 3.3.9 Scroll Net (IISCROLL)

#### Description

This module provides a windowing capability for building and displaying nets which outgrow the screen drawing limits. The 'FROM' and 'TO' x,y screen selections are input by the user indicating an x,y 'FROM' position on the net to be moved to an x,y 'TO' position on the screen. This coordinate translation is performed on the entire net and the net is removed from the screen and redrawn at its new position on the screen. The coordinate translation parameters are maintained internally so that the net can be continuously moved about.

#### Input

| | | |
|---|---|---|
| SCREEN SELECTION | - | The user inputs the 'FROM' and 'TO' x,y screen positions via the trackball facility. |
| ASSM | - | Entire ASSM net structure. |
| SCREEN MATRIX | - | Node positional and ASSM correlation data. |
| SCREEN LIMITS | - | Defines screen drawing area. |
| NODE DESCRIPTIONS | - | Node physical characteristics. |

#### Output

| | | |
|---|---|---|
| SCREEN MATRIX | - | Node positional and ASSM correlation data. |
| DISPLAY | - | Entire net is removed from screen and redrawn at new x,y position. |
| COORDINATE TRANSLATION PARAMETERS | - | See Section 3.3.1. |

#### Processing

A flow diagram of this module is shown in Figure 3-15. Selected processing steps are further clarified below.

| | | |
|---|---|---|
| [2] | - | A check is performed on the 'FROM' x,y screen position to insure that this position is, indeed, contained within the screen drawing area as defined by the SCREEN LIMITS. |

[6] - Coordinate translation parameters are computed from the x,y screen position input in Step 1 and 5.

Procedure References

The following correlates the functional processing steps shown in Figure 3-15 with the REVS procedures which perform the indicated processing.

[2] - IICHKNODESEL

[4] - IIMSGOUT

[7] - IICLEARSCREEN

[8] - IICLEARMATRIX

[9] - IIDISNET

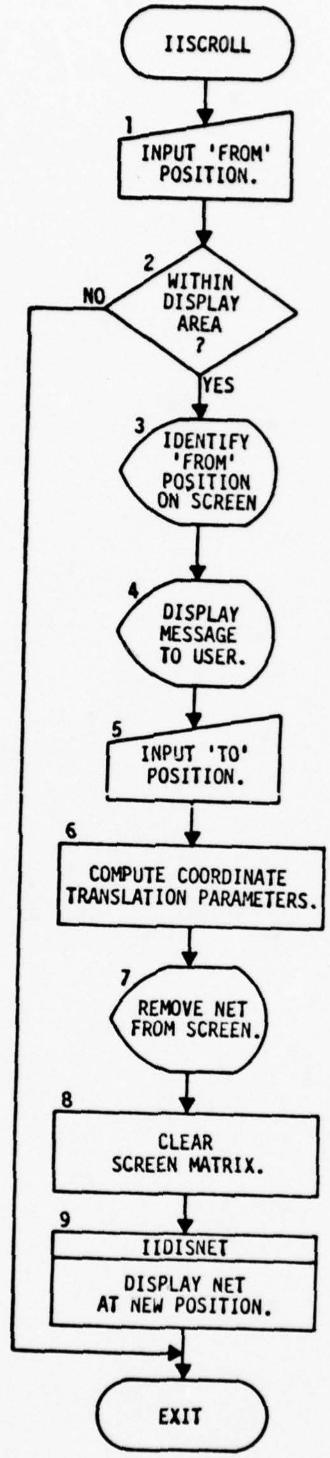



Figure 3-15 Scroll Net (IISCROLL)

### 3.3.10 Save Net (IISAVE)

#### Description

As an R-Net/Subnet is being built by the user, its structure is maintained in a temporary working area of the ASSM. If the structure is to be saved permanently in the ASSM, an explicit command must be issued by the user requesting such, as the structure will be lost when processing flow returns to the function level. No user inputs are required within this module level; however, structure analysis is performed prior to inserting the structure permanently in the ASSM and if errors exist, appropriate messages will be displayed to the user informing him that he must make appropriate structure changes before the structure can be put into the ASSM permanently.

#### Input

| | | |
|---|---|---|
| ASSM | - | The temporary structure of the net to be saved. |
| COORDINATE TRANSLATION PARAMETERS | - | See Section 3.3.1. |
| STRUCTURE STATUS FLAG | - | Error condition resulting from a structure analysis. |

#### Output

| | | |
|---|---|---|
| ASSM | - | Permanent net structure. |

#### Processing

The processing for this module is shown in the flow diagram of Figure 3-16. Selected processing steps are further described below.

| | | |
|---|---|---|
| [1] | - | The structure is checked for completeness and correctness. If it is found to be either incomplete or incorrect, it will not be saved. However, it is retained in temporary storage and the user may correct the indicated error and attempt to save again. |

Procedure References

The following correlates the functional processing steps shown in Figure 3-16 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | IIGRACHK, IICOMCHK, IILOOPCHK, IIANDORMATCHCHK, IICHKOTHERWISE |
| [4] | - | IICLEARSCREEN |
| [5] | - | IICLEARMATRIX |
| [6] | - | IIMSGOUT, IIDENTIFY |

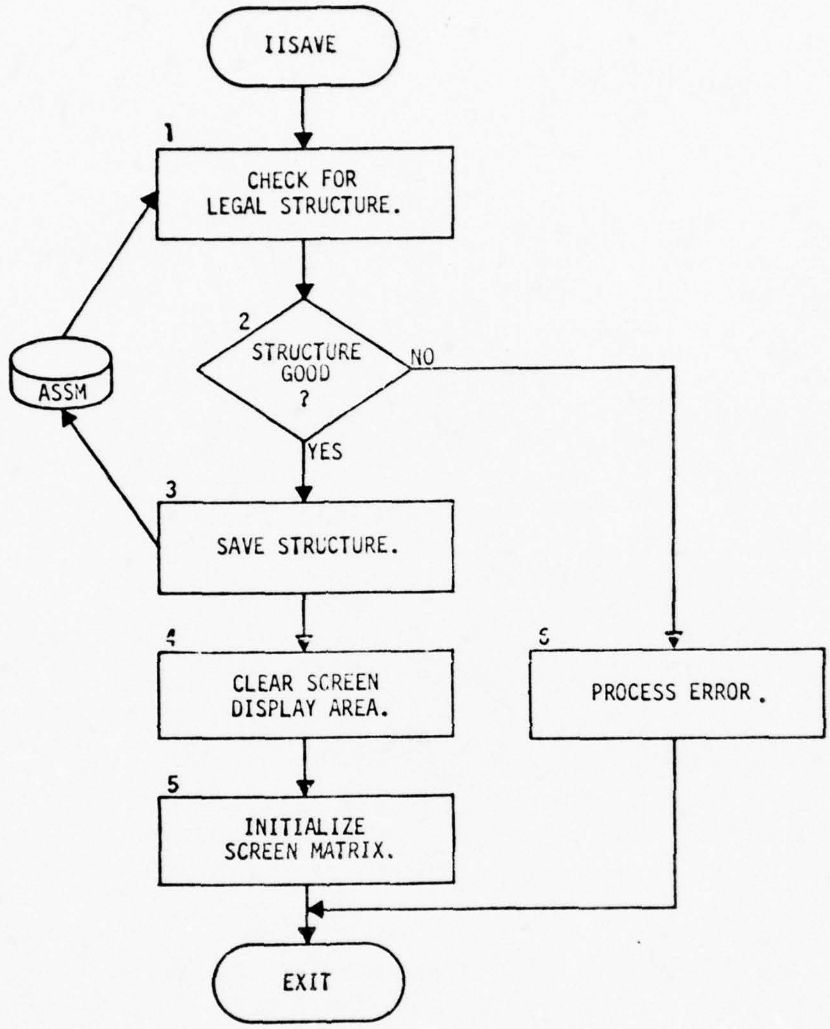



Figure 3-16 Save Net (IISAVE)

### 3.3.11 Zoom-Out On Net (IIZOOMOUT)

Description

As described in Section 3.3.9, the user may create a net structure, using the scroll net facility, which extends beyond the limits of the screen. IIZOOMOUT reduces the net structure such that it can be displayed in its entirety within the screen drawing area.

Input

| | | |
|---|---|---|
| ASSM | - | The entire net structure. |

Output

| | | |
|---|---|---|
| ASSM | - | Updated x,y positional data for the entire net. |
| DISPLAY | - | The entire net is displayed in a color-coded, zoomed-out mode. |

Local Data

| | | |
|---|---|---|
| COORDINATE TRANSFORMATION PARAMETERS | - | x,y transformation parameters required to translate from the zoomed-in net structure to the zoomed-out net structure. |

Processing

The processing performed by this module is shown in Figure 3-17. The following comments elaborate on the indicated processing for selected boxes.

| | | |
|---|---|---|
| [3] | - | The x,y translation and scaling parameters required to transform the net structure to a zoomed-out structure are computed. |

Procedure References

The following correlates the functional processing steps shown in Figure 3-17 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | IICLEARSCREEN |
| [2] | - | IICLEARMATRIX |
| [3] | - | IIFINDXY |
| [4] | - | IIDISZOUT |

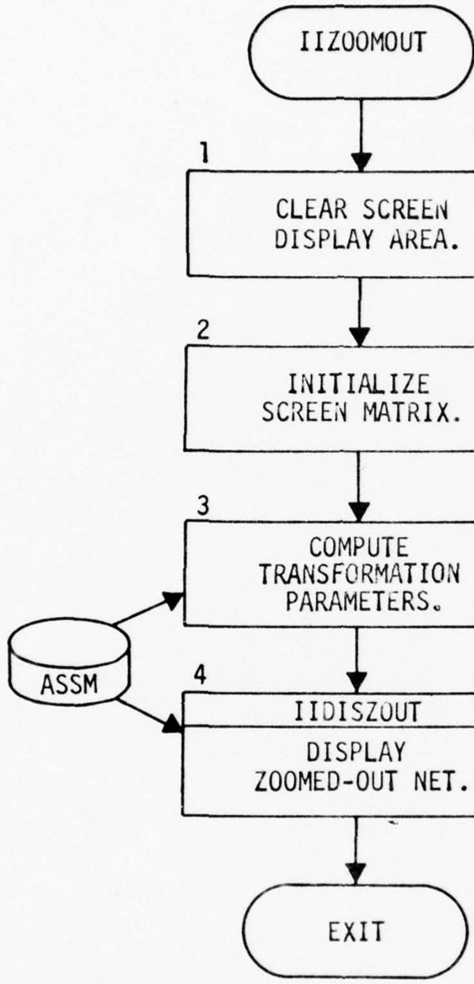



Figure 3-17 Zoom-Out On Net (IIZOOMOUT)

### 3.3.12 Zoom-In On Net (IIZOOMIN)

Description

As indicated in Section 3.3.9, the user may create a net structure which extends beyond the limits of the screen. A capability exists for him to zoom-out on such a structure so that the entire structure, regardless of size, can be totally contained within the screen display limits. When the net structure has been displayed in this 'zoomed-out' mode, the user may use the Zoom-In On Net module to select any point on the screen to be 'zoomed-in' on. The selected point is centered at the top of the screen and the net is displayed downward in a 'zoomed-in' mode to the screen limits.

Input

| | | |
|---|---|---|
| SCREEN SELECTION | - | The user selects an x,y point on the screen around which 'zooming-in' will take place. |
| ASSM | - | Entire net structure (zoomed-out). |
| SCREEN LIMITS | - | Defines screen drawing area. |
| NODE DESCRIPTIONS | - | Node physical descriptions. |

Output

| | | |
|---|---|---|
| ASSM | - | Screen positional data for entire net structure. |
| SCREEN MATRIX | - | Node positional and ASSM correlation data. |
| DISPLAY | - | The zoomed-out net is removed from the screen and the zoomed-in net is subsequently displayed. |

Processing

Figure 3-18 is a flow diagram of this module and the following comments apply to the indicated processing boxes.

| | | |
|---|---|---|
| [1] | - | A check is performed to insure that a zoomed-out net does, indeed, exist. |
| [2] | - | Coordinate translation parameters are computed and the net x,y positional data is transformed from the zoomed-out mode to the zoomed-in mode centered about the selected x,y screen positions. |

[5] - The net is displayed at its new position on the screen.

Procedure References

The following correlates the functional processing steps shown in Figure 3-18 with the REVS procedures which perform the indicated processing.

[3] - IICLEARMATRIX

[4] - IICLEARSCREEN

[5] - IIDISNET

[6] - IIMSGOUT

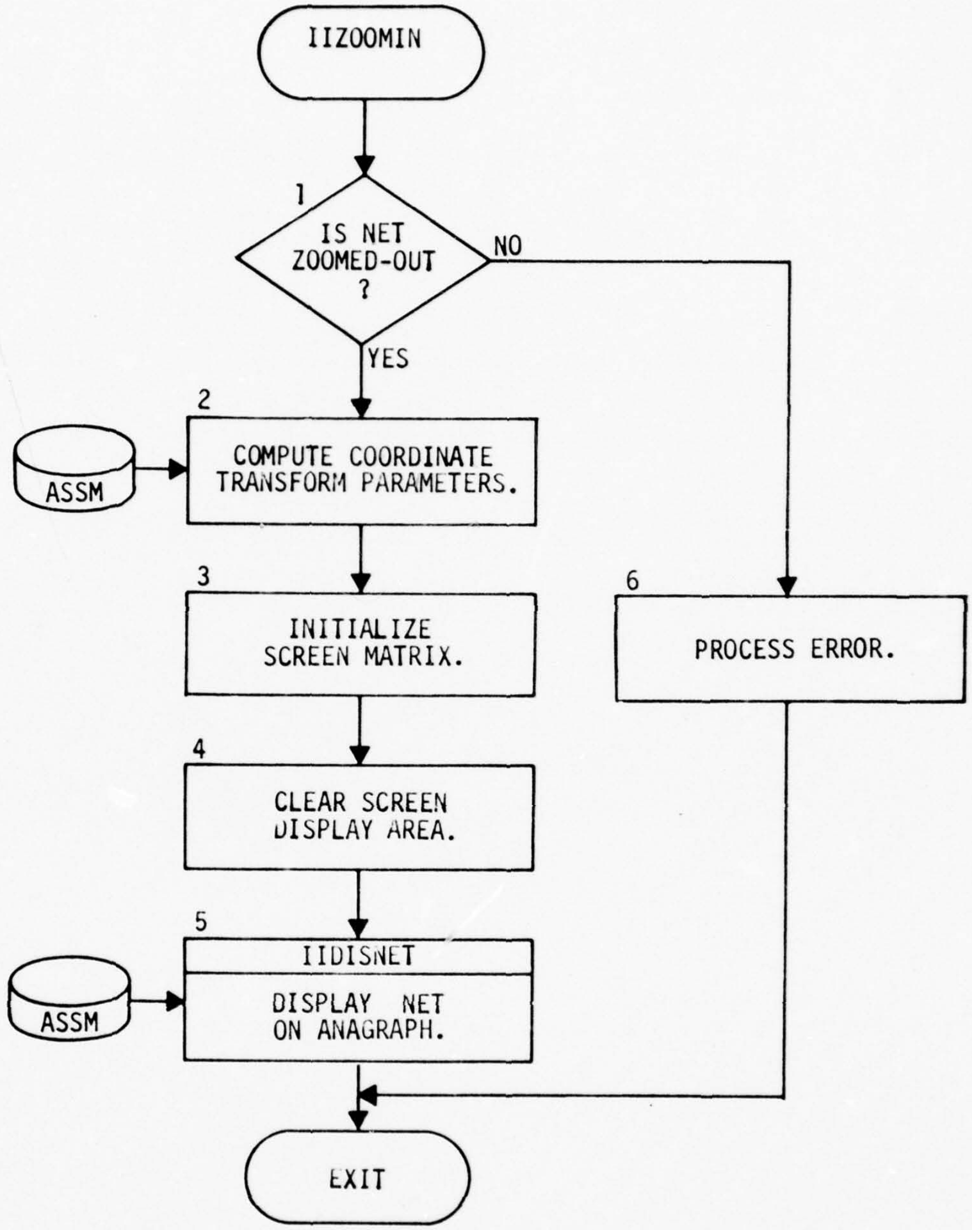



Figure 3-18 Zoom-In On Net (IIZOOMIN)

### 3.3.13 Generate CALCOMP Plot (IICALCOMP)

#### Description

This module generates a CALCOMP hard copy plot of any structure which has had graphics information entered through RNETGEN. Standard 8-1/2 by 11 document size output will be generated unless otherwise specified by the user. If necessary, the structure will be reduced in size such that the entire structure is contained within the selected document size.

#### Input

| | | |
|---|---|---|
| ASSM | - | The entire net structure and associated elements. |
| DOCUMENT SIZE | - | Width and height (inches) with default being 8-1/2 x 11. |

#### Output

| | | |
|---|---|---|
| NET DISPLAY | - | CALCOMP plot of current net structure. |

#### Processing

The following descriptions provide additional clarifying information for selected processing steps presented in Figure 3-19.

| | | |
|---|---|---|
| [1] | - | Standard document size of 8-1/2 x 11 inches is set up for CALCOMP output. |
| [2-5] | - | The user is given the option to accept standard document size or to specify optional width and height (inches). |
| [7] | - | This box represents a set of procedures (see Section 3.3.20) which produce the CALCOMP plot of a given structure. |

#### Procedure References

The following correlates the functional processing steps shown in Figure 3-19 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [2] | - | IIMSGOUT |
| [3] | - | IIMSGIN |
| [5] | - | IIVALIN |
| [7] | - | CCNET |
| [9, 10] | - | IIMSGOUT |

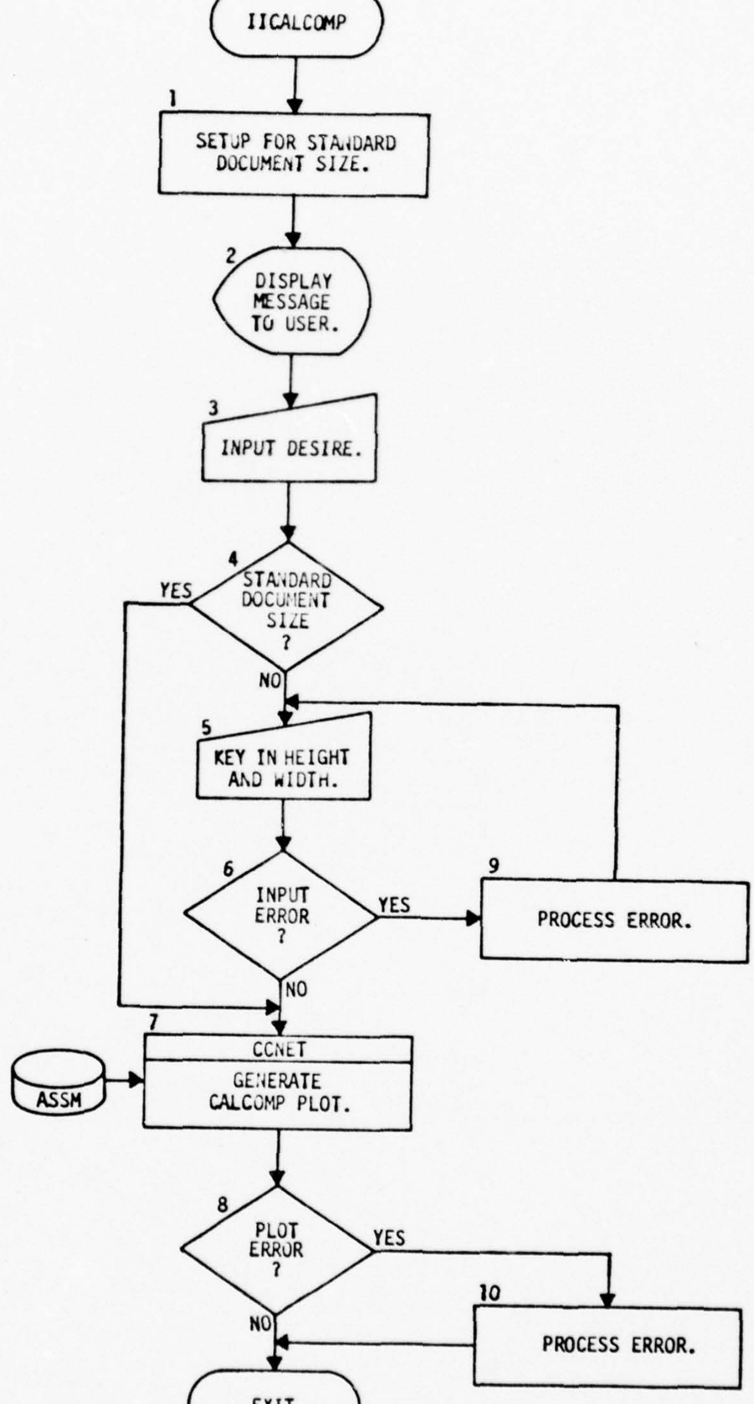



Figure 3-19 Generate CALCOMP Plot (IICALCOMP)

### 3.3.14 Set Color (IICOLOR)

Description

This module shown in Figure 3-20 determines the particular color selected from a menu of available colors to be used during net creation or modification. The selected color is used for displaying all subsequent nodes entered. The color attribute is added to the description of each node as it is entered and maintained in the ASSM. The color of an existing node may be changed by selecting the desired node via the trackball immediately following the color selection.

Input

COLOR MENU SELECTION - The user provides a color selection via the trackball input facility. The following colors are available in the menu:

- red
- green
- blue
- yellow
- purple
- turquoise
- white.

NODE SELECTION - The user selects the applicable node for which he wishes the selected color to apply.

Output

NODE COLOR - The selected color is maintained internally (color code is preset to turquoise).

Processing

The processing for this module is presented in the flow diagram of Figure 3-20.

Procedure Reference

The following correlates the functional processing steps shown in Figure 3-20 with the resulting REVS procedures which perform the indicated processing.

| | |
|---|---|
| [3] | - IISETCCOL |
| [4] | - IICMPMAT |
| [5] | - IIMATCHK |
| [6] | - IISETNODE |
| [7, 8] | - IINODIS |
| [9] | - IIMSGOUT |

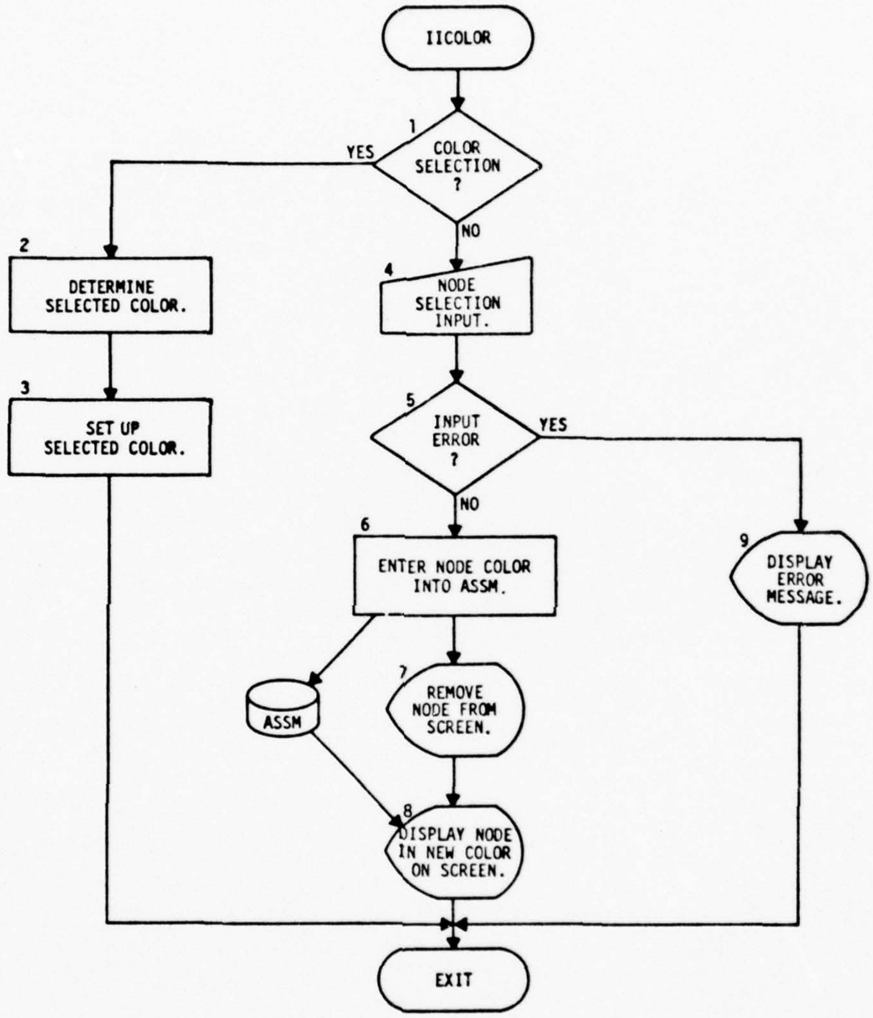



Figure 3-20 Set Color (IICOLOR)

### 3.3.15 Display Branch (IIDSPBRN)

Description

Upon input of the predecessor and successor node for the desired branch, this module will optionally display the branch ordinal value, if one exists, and the conditional expression, if one exists. The selected branch must be an 'OR/FOR' branch; otherwise, the input is rejected.

Input

| | | |
|---|---|---|
| NODE POSITIONS | - | This input is provided by the user via trackball selections of the predecessor/successor nodes. |
| SCREEN MATRIX | - | Screen positional and ASSM correlation matrix. |
| ASSM | - | Structure data. |
| SCREEN LIMITS | - | Defines screen drawing area. |

Output

| | | |
|---|---|---|
| DISPLAY | - | Display of ordinal value, if applicable, and conditional expression. |

Processing

Figure 3-21 presents a flow diagram of the processing steps within this module. Following is further clarifying information for the indicated processing steps.

| | | |
|---|---|---|
| [2] | - | Checks are made to insure that the selected node does indeed exist and that it is either an 'OR' or a 'FOR EACH' node. |
| [4] | - | Checks are made to determine the existence of the selected node and to insure that it is indeed a successor to the previously selected node. |
| [5] | - | The user is given the option to have the ordinal displayed for the indicated branch, if indeed, one exists. |
| [6] | - | Again, the user is given the option to have the conditional expression displayed, if one exists. |

Procedure References

The following correlates the functional processing steps shown in Figure 3-21 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | IICMPMAT |
| [2] | - | IIMATCHK |
| [3] | - | IICMPMAT |
| [4] | - | IICHKNODESEL, IIMATCHK |
| [5] | - | IIMSGOUT, IIMSGIN |
| [6] | - | IIMSGOUT, IIMSGIN |
| [7] | - | IIMSGOUT |

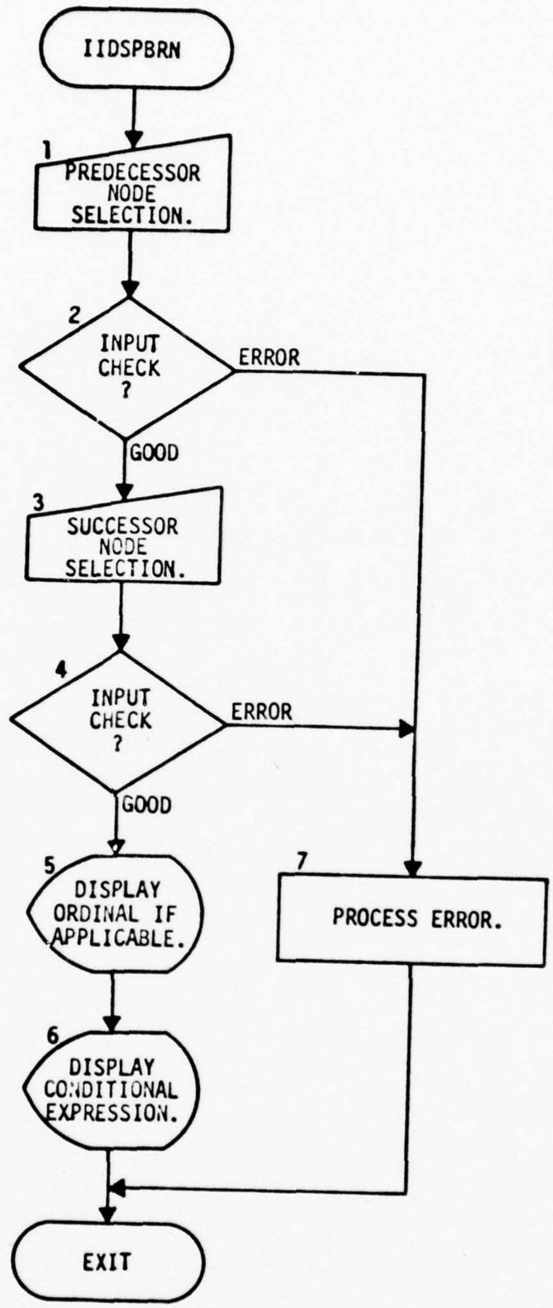



Figure 3-21 Display Branch (IIDSPBRN)

### 3.3.16 Display Node (IIDSNODE)

Description

Upon input of the desired node, this module will display the full RSL name of the element associated with the node, if applicable.

Input

| | | |
|---|---|---|
| NODE POSITION | - | The input is provided by the user via a trackball selection of the desired node. |
| SCREEN MATRIX | - | Screen positional and ASSM correlation matrix. |
| ASSM | - | Structure related data. |
| SCREEN LIMITS | - | Defines screen drawing area. |

Output

| | | |
|---|---|---|
| DISPLAY | - | RSL element name of associated element, if applicable. |

Processing

Figure 3-22 presents a flow diagram of the processing steps within this module. Following is further clarifying information for the indicated processing steps.

| | | |
|---|---|---|
| [2] | - | Checks are made to insure that the selected node does indeed exist on the screen. |
| [3] | - | A further check is made to determine whether the selected node is associated to an element in the ASSM. |
| [4] | - | The full RSL element name is retrieved from the ASSM and displayed on the screen. |

Procedure References

The following correlates the functional processing steps of Figure 3-22 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | IICMPMAT |
| [2] | - | IICHKNODESEL |
| [4, 5] | - | IIMSGOUT |

Procedure References

The following correlates the functional processing steps of Figure 3-22 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | IICMPMAT |
| [2] | - | IICHKNODESEL |
| [4, 5] | - | IIMSGOUT |

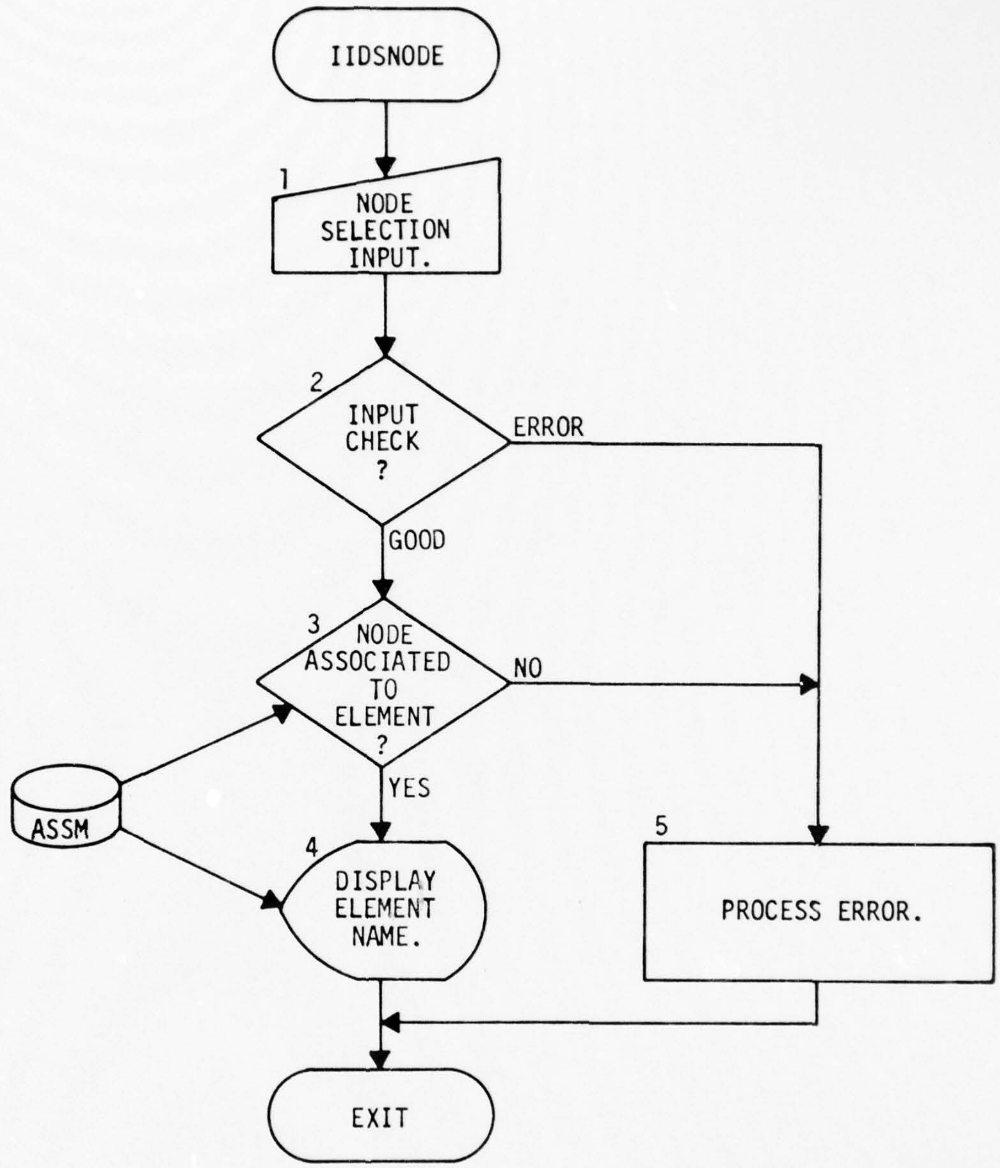



Figure 3-22 Display Node (IIDSNODE)

### 3.3.17 Display Net (IIDISNET)

Description

This module is used to display an R-Net/Subnet in the 'zoomed-in' mode. Each node on the net is interrogated to determine if it will fit within the screen display limits and if it does not fit, the node is ignored and processing continues at the next node.

Input

| | | |
|---|---|---|
| ASSM | - | The entire net structure. |
| SCREEN LIMITS | - | Defines screen drawing area. |
| NODE DESCRIPTIONS | - | Node physical descriptions. |
| NODE COLOR | - | Selected by the user. |

Output

| | | |
|---|---|---|
| DISPLAY | - | Net display on the screen. |
| SCREEN MATRIX | - | Node positional and ASSM correlation data. |

Processing

The processing for this module is shown in Figure 3-23. The following provides additional processing descriptions for the indicated boxes.

| | | |
|---|---|---|
| [1] | - | Node record data and successor/predecessor data is retrieved from the ASSM. |
| [2] | - | The x,y positional data is translated, if required, using the coordinate translational parameters computed by the Scroll Net module. |
| [3] | - | A check is performed to insure that the node will fit on the screen. If the node does not fit on the screen, processing is continued at the next node on the net. |
| [4] | - | The appropriate node color is set up for subsequent display. |
| [5] | - | The node is displayed on the screen at the appropriate x,y position. |

[6] - The internal screen matrix is updated to reflect the node screen position.

[7] - All directed arcs from/to this node are displayed and processing returns to step [1] until all nodes on the net have been processed.

Procedure References

The following correlates the functional processing steps shown in Figure 3-23 with the REVS procedures which perform the indicated processing.

[3] - IICMPMAT, IICHKNODESEL

[4] - IISETNODE

[5] - IINODIS, IIDISNAM

[7] - IIDSPLINES

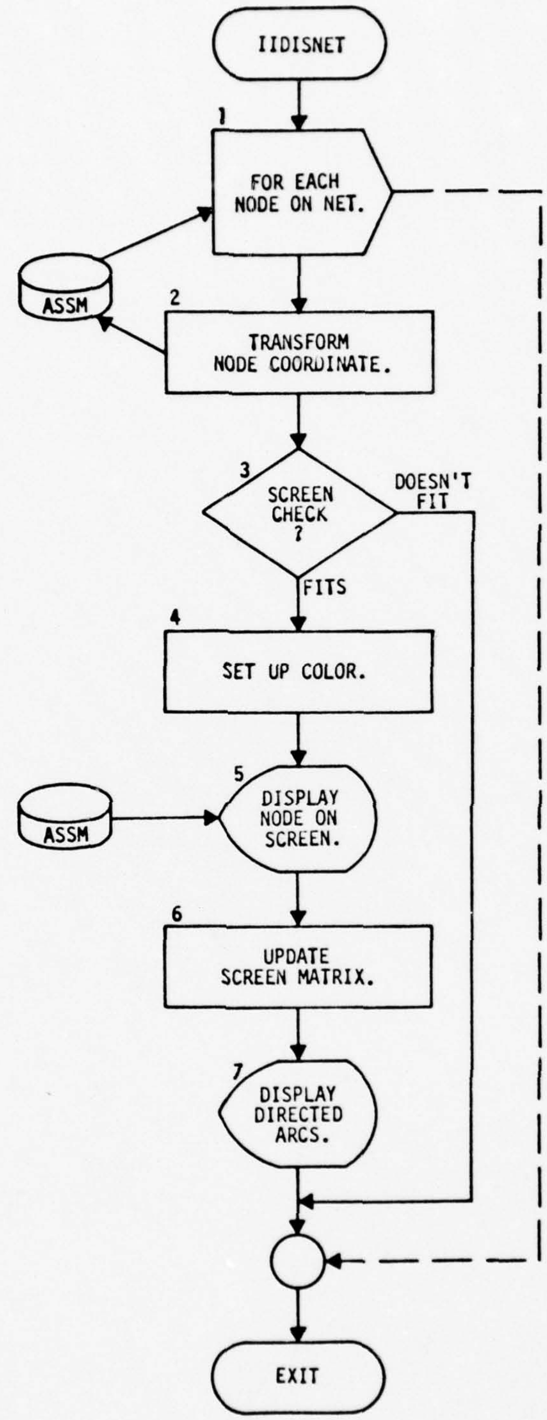



Figure 3-23 Display Net (IIDISNET)

## 3.3.18 Display Zoomed-Out Net (IIDISZOUT)

### Description

This module is used to display a color-coded miniature display of the current structure. This provides a capability for displaying the entire structure, regardless of size, within the structure display area of the CRT. The x,y scaling factors to translate from the nominal (zoomed-in) structure mode to the zoomed-out mode are computed and all connecting arcs on the structure are first displayed from node-center to node-center. The nodes are subsequently displayed in color-coded form.

### Input

| | | |
|---|---|---|
| ASSM | - | The entire net structure. |
| SCREEN LIMITS | - | Defines screen drawing area. |
| NODE CHARACTERISTICS | - | Color codes and physical descriptions. |

### Output

| | | |
|---|---|---|
| DISPLAY | - | Miniature color-coded display of structure. |
| SCREEN MATRIX | - | Zeroed out. |

### Processing

Figure 3-24 presents a flow diagram for the processing in this module.

### Procedure Reference

The following correlates the functional processing steps shown in Figure 3-24 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [8] | - | IISETNODE |

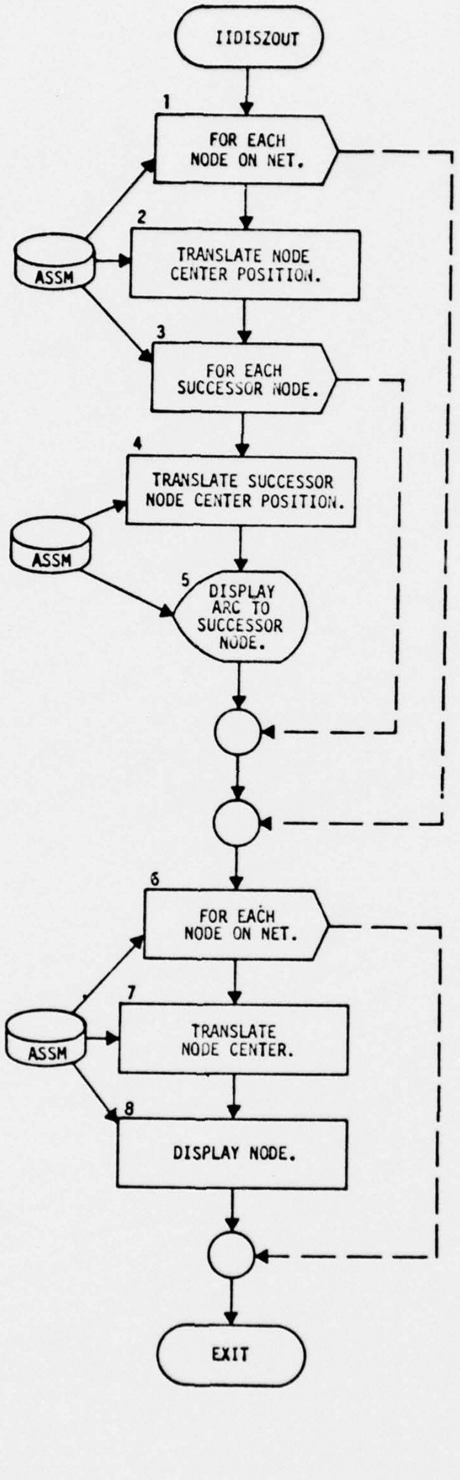



Figure 3-24 Display Zoomed-Out Net (IIDISZOUT)

### 3.3.19 Menu Read (IIMENUREAD)

Description

This module provides menu selection capability via the trackball input facility. The selected x,y screen coordinate is translated to a menu line entry. Appropriate indicators are set and control is returned to the function level.

Input

| | | |
|---|---|---|
| MENU LIMITS | - | Defines menu limits. |
| MENU LINES | - | Defines menu line entries. |
| SCREEN SELECTION | - | x,y screen coordinate value as input via the trackball facility. |

Output

| | | |
|---|---|---|
| MENU SELECTION | - | Selected menu line entry. |

Processing

The processing for this module is presented in Figure 3-25. The following comments provide additional clarifying information for the indicated processing boxes.

| | | |
|---|---|---|
| [3] | - | A determination is made as to whether the selected x,y screen position indicates a menu line entry. |
| [4] | - | A check is made to insure that the selected menu entry is indeed legal. |
| [5] | - | If a structure type was selected in the menu, a determination is made as to what type (i.e., RNET, SUBNET, VALPATH). |
| [6] | - | If a node type was selected in the menu, a determination is made as to what type (i.e., ALPHA, EVENT, OR, AND, etc.). |
| [7] | - | If neither a structure type or node type was selected, then the appropriate menu entry is determined and so indicated (i.e., move node, scroll, net, etc.). |

Procedure References

The following correlates the functional processing steps shown in Figure 3-25 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [10] | - | IICHKNODESEL |
| [11, 12] | - | IIMSGOUT |

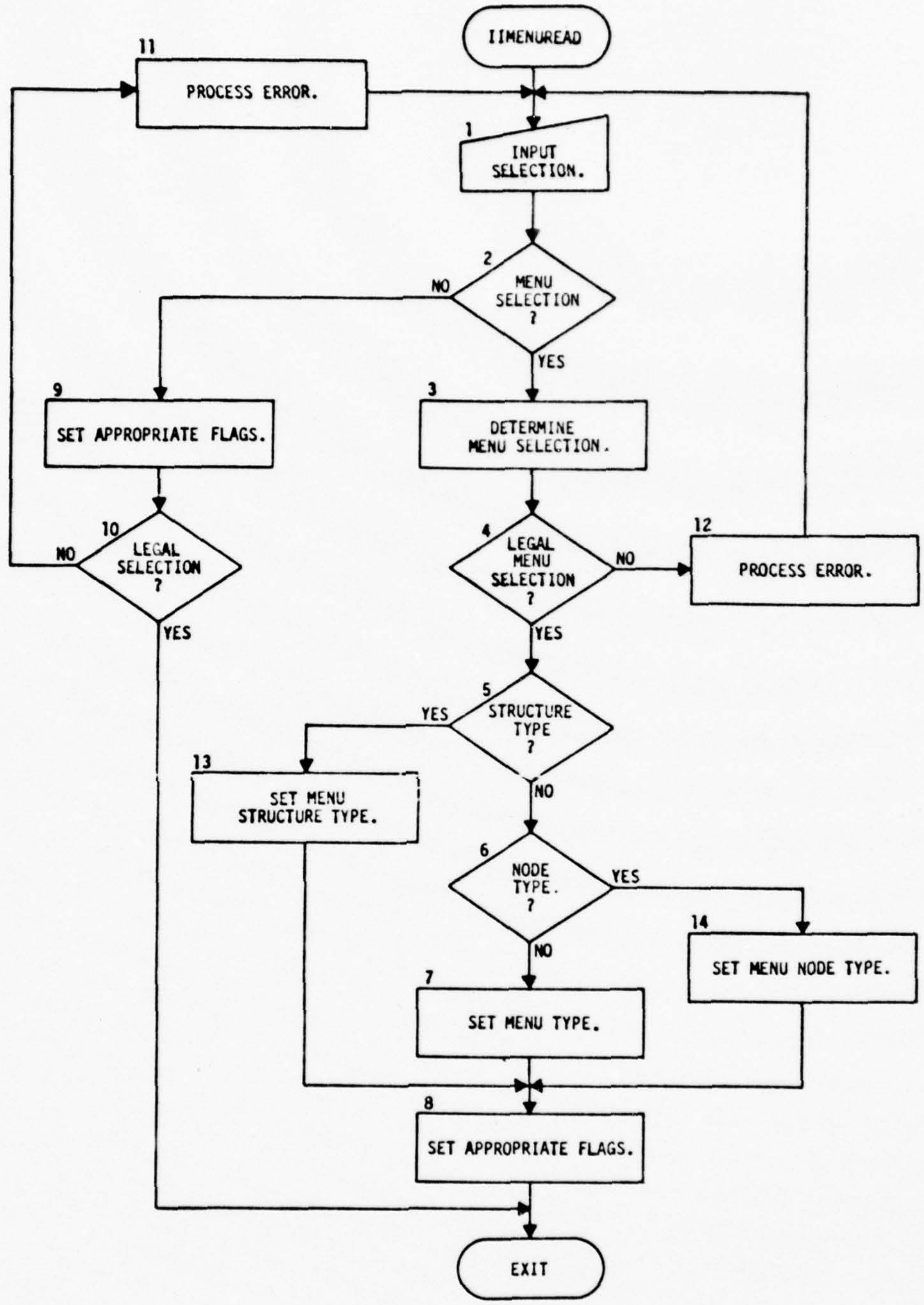



Figure 3-25 Menu Read (IIMENUREAD)

### 3.3.20 CALCOMP Net Display (CCNET)

#### Description

This module will plot a selected structure in the ASSM which has associated graphics coordinate data. Its output is recorded on a CALCOMP compatible tape for plotting on 30-inch paper. The plot size is provided via the argument list along with the ASSM pointer to the element to which the structure is attached.

#### Input

| | | |
|---|---|---|
| ASSM | - | The entire net structure and associated elements. |
| DOCUMENT SIZE | - | Desired width and height (inches) of the plotted output. |
| FLAG | - | First pass flag. |

#### Output

| | | |
|---|---|---|
| DISPLAY | - | CALCOMP plots of the selected structure. |

#### Processing

The following information is presented for clarification of selected processing steps appearing in Figure 3-26.

| | | |
|---|---|---|
| [1] | - | If the selected ASSM element has no associated structure, an error flag is set and control is returned to the calling program. |
| [2] | - | All required ASSM pointers to RSL element types are retrieved for subsequent use. |
| [3, 9] | - | If the selected structure is not currently in temporary storage it is moved there. |
| [6] | - | x,y scale factors to translate Anagraph coordinate units to CALCOMP coordinate units such that the entire structure will be contained within the requested document size are computed. |

[7, 8] - If this is the first execution of CCNET, the CALCOMP tape is initialized and the TRW logo frame is generated.

[10] - If the structure contains no associated graphics data, the graphics data will be generated automatically and entered in the ASSM.

[16, 21] - If a node is associated to an ASSM element, the element is retrieved and displayed at the node on the structure.

[20, 22] - If a conditional expression exists on a node branch, 'OR/FOR EACH' node, then the branch is numbered on the structure and the associated ordinal and conditional expression is entered in the branch legend of the structure.

[24] - If the structure was not already in temporary storage upon entry into CCNET, then it is removed from temporary storage.

Procedure References

The following correlates the functional processing steps shown in Figure 3-26 with the REVS procedures which perform the indicated processing.

[6] - CCFINDXY, CCAMSN, CCAMAX

[8] - CCTRWLOGO

[10] - CCAUTOPLT

[13] - CCSETNODE

[21] - CCPUTEXT

[22] - CCPRINTNOBR, CCPRINTTBBR

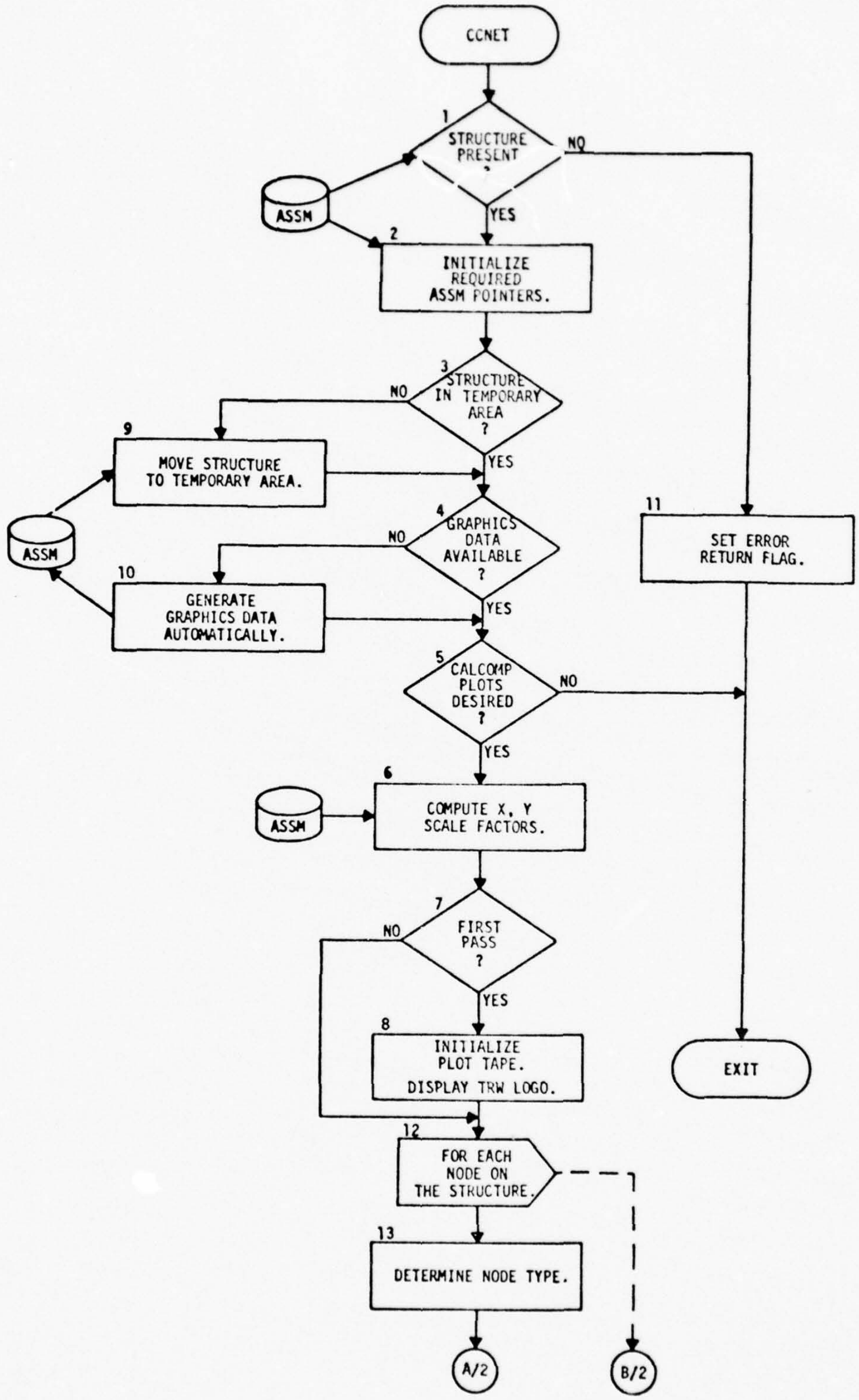



Figure 3-26 CALCOMP Net Display (CCNET)

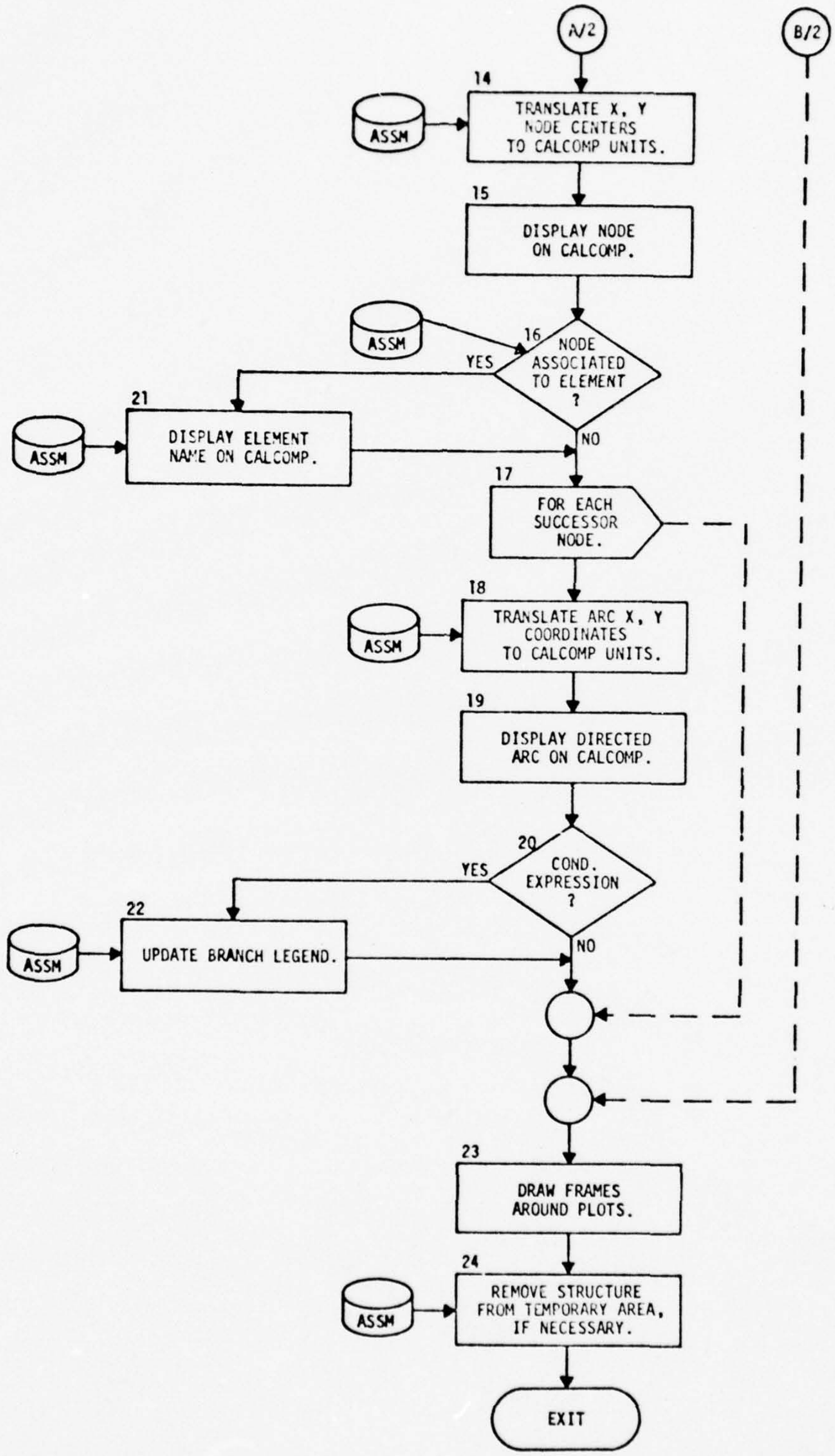



Figure 3-26 CALCOMP Net Display (CCNET) (Continued)

### 3.3.21 Autoplot (IIAUTOPLOT)

Description

This module provides the capability for the automatic generation of graphics coordinate data for the current structure. All nodes on the structure will also have their color changed to the currently selected color in the menu. The resulting structure will in general, have a neater and more pleasing appearance than one which is drawn manually via RNETGEN.

Input

| | | |
|---|---|---|
| ASSM | - | The entire net structure and associated elements |
| COLOR | - | Current color as indicated via the menu. |

Output

| | | |
|---|---|---|
| SCREEN MATRIX | - | Updated screen matrix. |
| ASSM | - | Graphics coordinate and color data for each node on the current structure. |
| DISPLAY | - | The structure is displayed in either its zoomed-out or zoomed-in mode. |

Processing

The following information is presented for clarification of selected processing steps appearing in Figure 3-26.1.

| | | |
|---|---|---|
| [2] | - | The color on all nodes in the structure is set to blank. This is required by the automatic graphics coordinate data generator program. |
| [3] | - | This procedure will automatically generate graphics coordinates for each node on the structure. |
| [4] | - | All nodes on the structure is given the color of that specified on the menu. |
| [6, 7, 8, 9] | - | The user selects, via the trackball, to display the structure in either its zoomed-out or its zoomed-in mode. |

Procedure References

| | | |
|---|---|---|
| [1] | - | IICLEARSCREEN |
| [2] | - | IIREMGRAPH |
| [3] | - | XXCNET |
| [4] | - | IIREMGRAPH |
| [5] | - | IIMATCLEAR |
| [8] | - | IIZOOMOUT |
| [9] | - | IIDISNET |

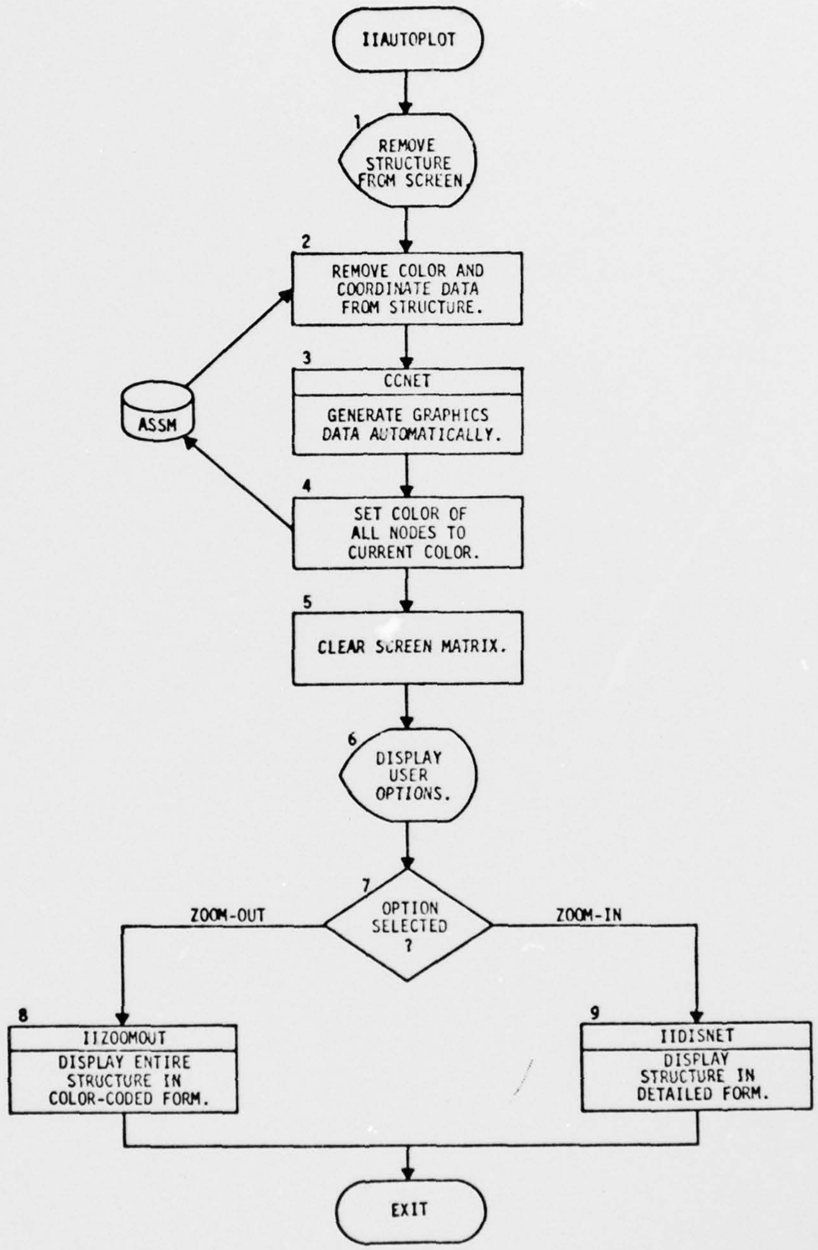



Figure 3-26.1 Autoplot (IIAUTOPLOT)

### 3.3.22 Move Subtree (IIMVSUBTREE)

Description

This module allows the user to move portions of the currently displayed structure by selecting an existing node on the structure and specifying a position on the screen to which the selected node is to be moved. The selected node and all nodes on the structure below it are moved accordingly.

Input

| | | |
|---|---|---|
| ASSM | - | The entire net structure and associated elements. |
| NODE SELECTION | - | The user specifies the subtree to be moved via a node selection using the trackball. |
| SCREEN MATRIX | - | Node positional and ASSM correlation data. |
| SCREEN LIMITS | - | Defines screen drawing area. |
| NODE DESCRIPTIONS | - | Node physical characteristics |
| SCREEN SELECTION | - | The user specifies the x,y 'to' position on the screen to which the node will be moved. |

Output

| | | |
|---|---|---|
| ASSM | - | Coordinate data for selected node and associated successor/predecessor nodes in the subtree which was moved. |
| DISPLAY | - | Removal of selected subtree at current screen position and display of subtree at newly selected screen position. |
| SCREEN MATRIX | - | Updated screen matrix. |

Processing

The following information is presented for clarification of selected processing steps appearing in Figure 3-26.2.

| | | |
|---|---|---|
| [1] | - | The user identifies a subtree on the currently displayed structure by selecting the leading node of the desired subtree to be moved. |

[2] - A validity check on input is performed. First, to determine whether the selected x,y position is within the screen drawing limits; and, secondly, to determine if a node exists at the selected x,y position.

[3] - The user inputs the desired x,y screen position to which the node selected in Step [1] is to be moved.

[4] - An input check is performed to insure that the selected x,y position is within the screen limits.

[7] - The identified subtree structure is removed from the screen and its new coordinates are computed and entered into ASSM.

[10] - The identified subtree structure is displayed at its new position on the screen.

Procedure References

The following correlates the functional processing steps shown in Figure 3-26.2 with the REVS procedures which perform the indicated processing.

[1] - IICMPMAT

[2] - IIMATCHK

[4] - IICHKNODESEL

[6] - IIDETNOD

[7] - IIPROCBRNCH

[9] - IIDETNOD

[10] - IIPROCBRNCH

[11] - IIMSGOUT

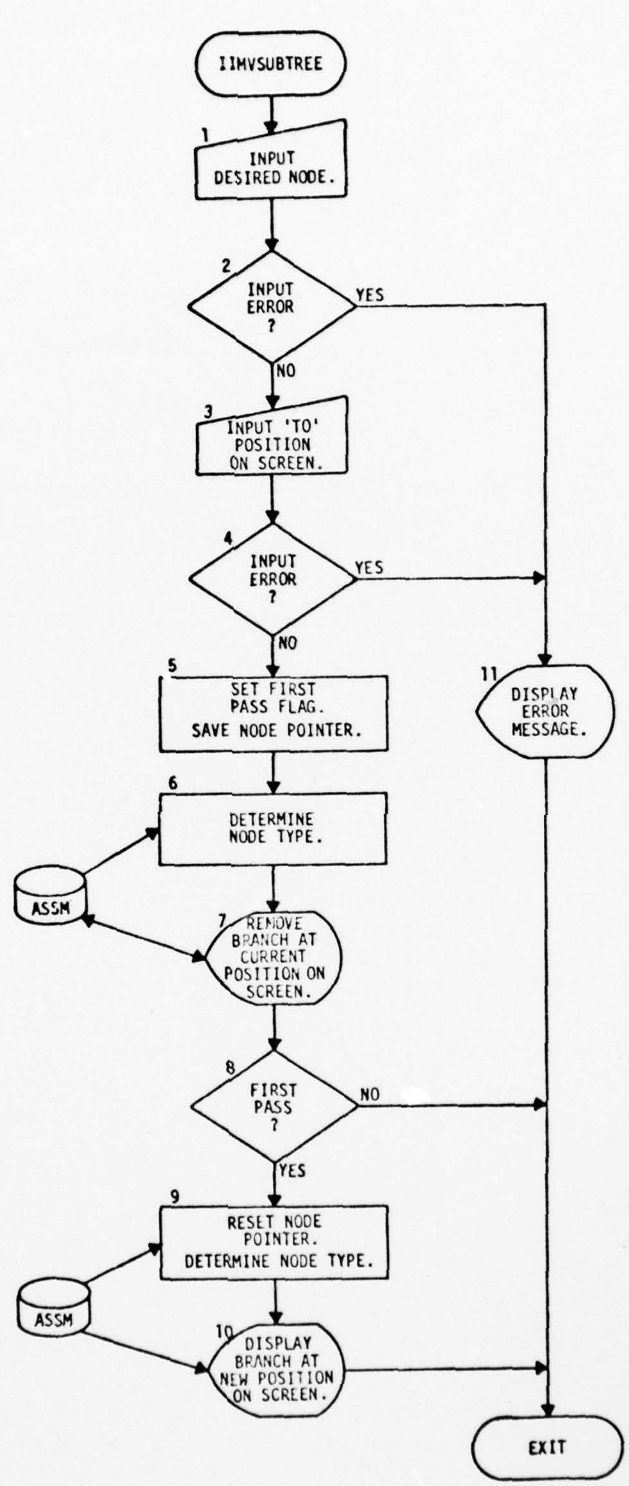



Figure 3-26.2 Move Subtree (IIMVSUBTREE)

## 3.4 REQUIREMENTS ANALYSIS AND DATA EXTRACTION (RADX)

### Description

The Requirements Analysis and Data Extraction function provides a variety of capabilities to aid in the development of requirements specifications. These include a generalized ASSM query and data extraction facility, an RSL documentation capability, and a static analysis capability to identify simple and complex anomalies in the requirements specifications.

The query and data extraction facility uses the concept of a SET, a collection of ASSM elements, to provide flexible and powerful user control for the interrogation and documentation of the contents of the ASSM. Documentation is selectively generated in either RSL text or a hierarchical map form. There are two types of SETs -- predefined and user defined. The predefined SETs are:

- The universal set which is referred to as ALL_SET contains all the elements in the ASSM.
- Element-type sets which are referred to by the element-type name contain all the elements that are of the named element-type.
- Element sets which are referred to by the element name contain the named element.

A user defined set is a collection of elements defined by the user in one of the following ways:

- The elements in a list of SETs
- The logical combination of two SETs
- The elements in a SET that satisfy a qualification criterion.

The RSL documentation capability allows all or part of the currently legal definition of RSL to be displayed. This includes the description of element-types, relations, and attributes.

The static analysis capability diagnoses anomalies in the flow-oriented portion of the requirements. Such things as loops in a data or structure

hierarchy, illegal combinations of relationships or attributes, and data flow errors are identified.

Input

| | | |
|---|---|---|
| USER RCL | - | RADX control statement that defines operations to be performed. |
| ASSM | - | Any part of the ASSM can be retrieved by RADX. |
| TYPE ACTIVATION | - | In addition to the normal interface with the user, RADX interfaces with the SIMGEN function (see Section 3.5) to perform static analysis and data collection for that function. |

Output

| | | |
|---|---|---|
| DIAGNOSTIC MESSAGE | - | Input control statement not processed due to error during statement translation. |

Other outputs from RADX are made from the various modules that compose RADX.

Local Data

The following information is generated by translating user RCL.

| | | |
|---|---|---|
| TYPE_COMMAND | - | Indicates operation to be performed by RADX. Operations are: DEFINE_SET; QUALIFY_SET; COMBINE_SETS; DEFINE_HIERARCHY; DEFINE_APPEND; LIST_SET; LIST_HIERARCHY; LIST_RSL; ANALYZE; LIST_PERMISSION; PLOT |
| WIDTH | - | Width of PLOT. |
| HEIGHT | - | Height of PLOT. |
| PERMISSION_ID | - | Identifier of the CONTROL_PERMISSION to be displayed. |
| MEMBER_LIST | - | List of SETs for defining a new set. |
| NEW_SET_NAME | - | The name of the new set that results from performing a DEFINE_SET, QUALIFY_SET or COMBINE_SET operation. |
| TYPE_QUALIFY | - | Indicates the technique used to qualify the members of an existing set to form |

| | | |
|---|---|---|
| | | a new set. Legal techniques are BY_ATTRIBUTE, BY_RELATION, and BY_HIERARCHY. |
| QUALIFYING_ATTRIBUTE | - | Attribute used to qualify a set. |
| QUALIFYING_VALUE | - | The value of an attribute used to qualify a set. |
| QUALIFYING_RELATION | - | Relation used to qualify a set. |
| QUALIFYING_OBJECT_SET | - | Collection of elements used when qualifying BY_RELATION. |
| FIRST_INDEPENDENT_SET | - | The first set of a COMBINE_SETS operation. |
| SECOND_INDEPENDENT_SET | - | The second set of a COMBINE_SETS operation. |
| TYPE_COMBINATION | - | Indicates how sets are to be logically combined. The combination can be UNION, INTERSECTION, or DIFFERENCE. |
| HIERARCHY_NAME | - | The name of a user defined hierarchy. |
| HIERARCHY_ENTRIES | - | A list of one or more triplets which define how to trace direct and indirect relationships between elements in the ASSM. The triplet is (SUBJECT_TYPE, BINDING_RELATION, OBJECT_TYPE). |
| SELECTED_APPEND_TYPE | - | Indicates the element type that has been selected to have its APPEND_OPTION changed. |
| APPEND_ITEM_LIST | - | List of associated information such as relations and attributes to be displayed when an element of a particular type is displayed. |
| INDEPENDENT_SET | - | Collection of elements to be listed, qualified, or analyzed. |
| SELECTED_HIERARCHY | - | Identifier of a previously defined hierarchy that is to be used for listing or qualifying a set. |
| HIER_DISPLAY_FORM | - | Indicates format to be used for displaying hierarchy. Legal values are MAP, SEQUENCE, and GROUP. |

RSL_LIST_OPTION - Indicates what portion of RSL definition has been selected to be displayed. Legal selections are ALL, TYPES, RELATIONS, ATTRIBUTES, SUMMARY, ONE_TYPE, ONE_RELATION, ONE_ATTRIBUTE, and ONE_TYPE_SUMMARY.

DATA_FLOW_OPTION - Indicates that option to perform data flow analysis has been selected.

Figure 3-27 illustrates the data structures used by RADX for the management of SETs and their related information. The following further explains these data structures:

SET_LIST_ARRAY - An entry is made into this array for each set known to RADX. The entry contains the location of the SET_DESCRIPTION_RECORD. The first entry contains the location of ALL_SET which is the universal set of all elements in the ASSM. Entries 2 through n+1 contain the locations for the n predefined sets. The remainder of the entries are reserved for user defined sets.

SET_DESCRIPTION_RECORD - Contains the name of the set, the location of the first APPEND_OPTION_RECORD, and the location of the first SET_MEMBER_RECORD.

APPEND_OPTION_RECORD - Contains one type of information to be displayed when an element that is a subset of the owning SET_DESCRIPTION_RECORD is displayed.

SET_MEMBER_RECORD - Contains indexes into the ASSM_ELEMENT_ARRAY which identify the members of the set.

ASSM_ELEMENT_ARRAY - An entry, which contains the location of the element, is made in this array for each element in the ASSM. The array is ordered alphabetically by element type.

## Processing

Figure 3-28 contains the functional flow diagram for this module.

## Procedure References

The following correlates the functional processing elements in Figure 3-28 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | QQINIT |
| [3, 5] | - | QQTRDXM |
| [10-20] | - | QQDXM |
| [21, 22] | - | QQDXM, QQLISTSET |

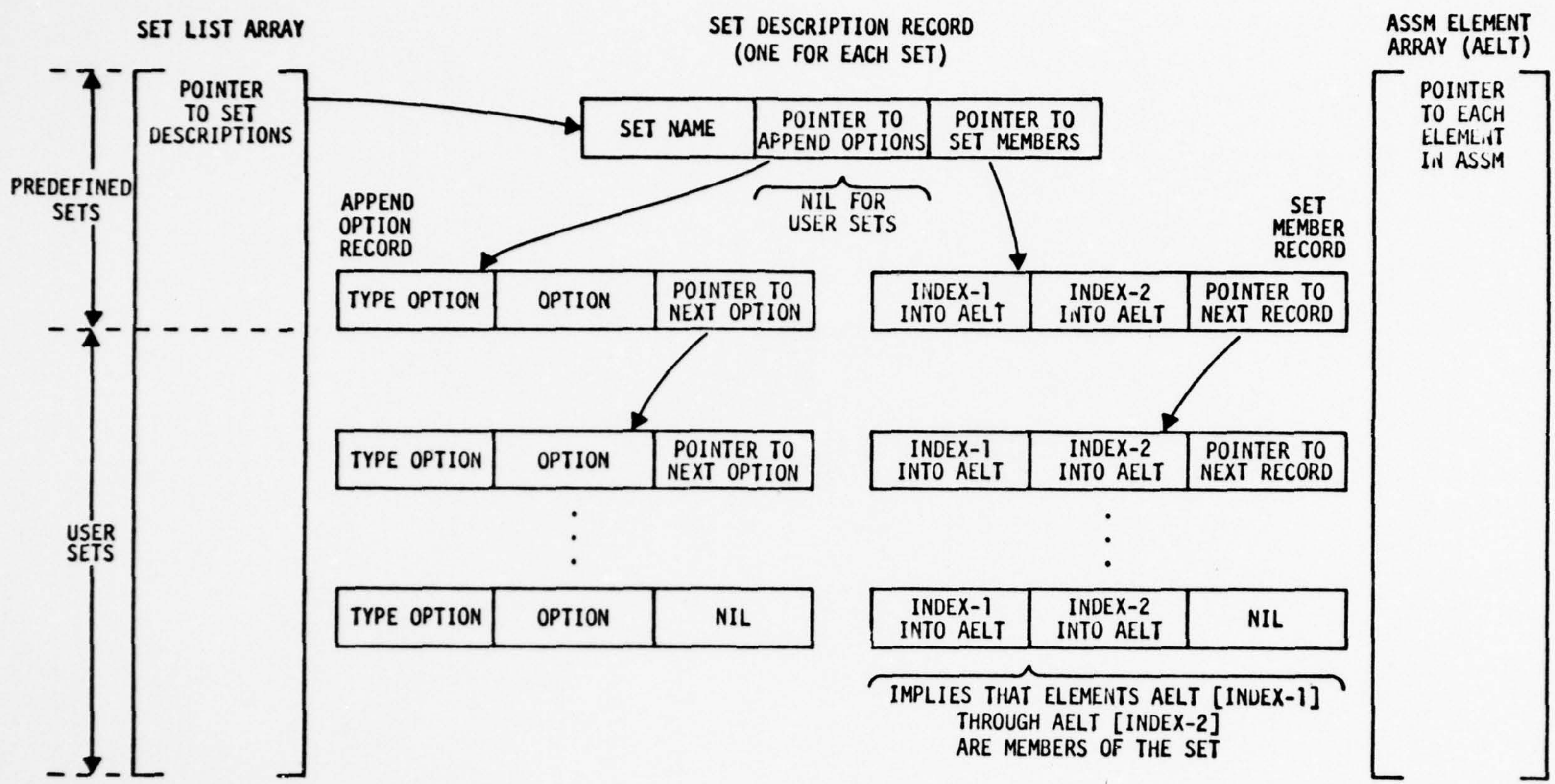



Figure 3-27 RADX Data Structures for Set Management

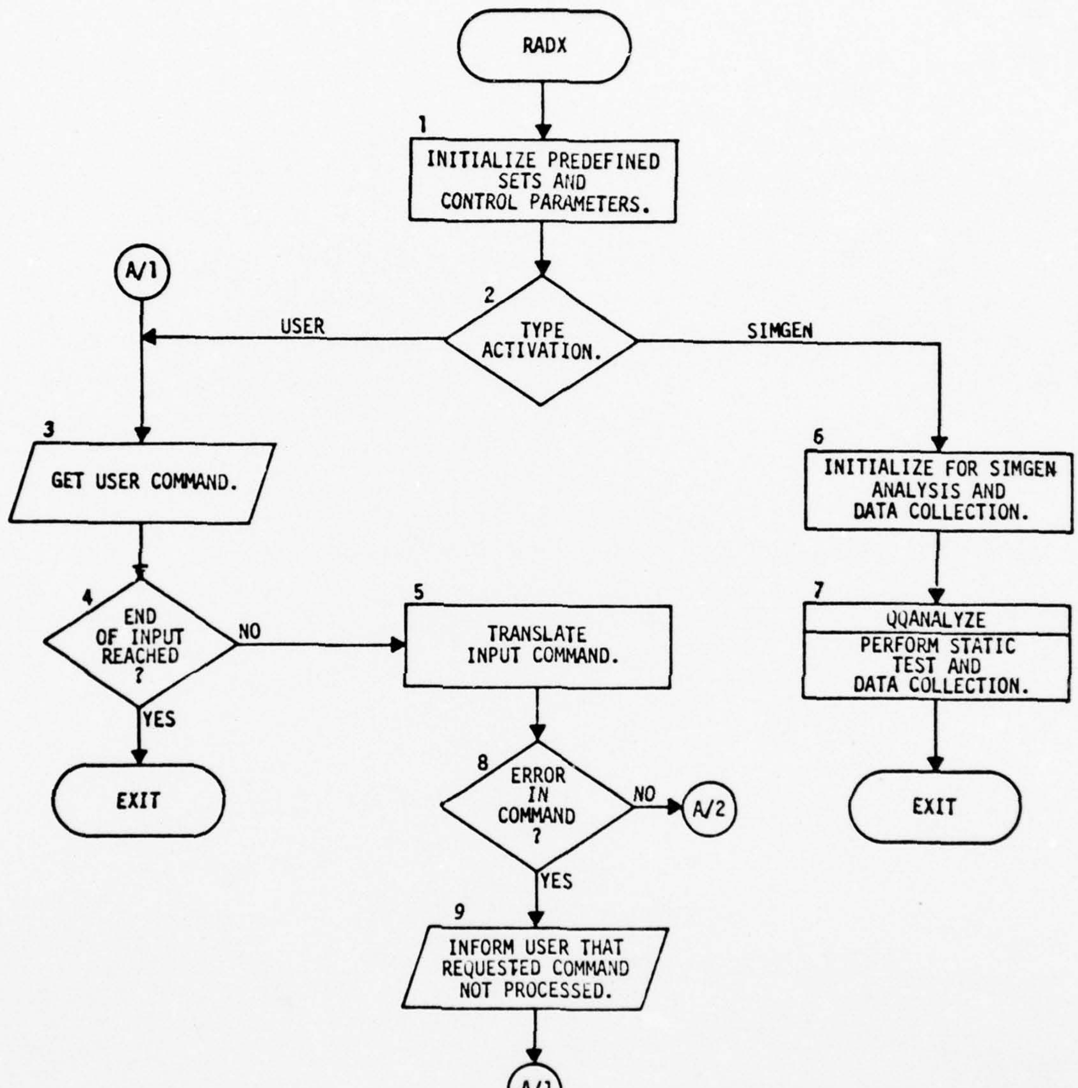



Figure 3-28 Requirements Analysis and Data Extraction (RADX)

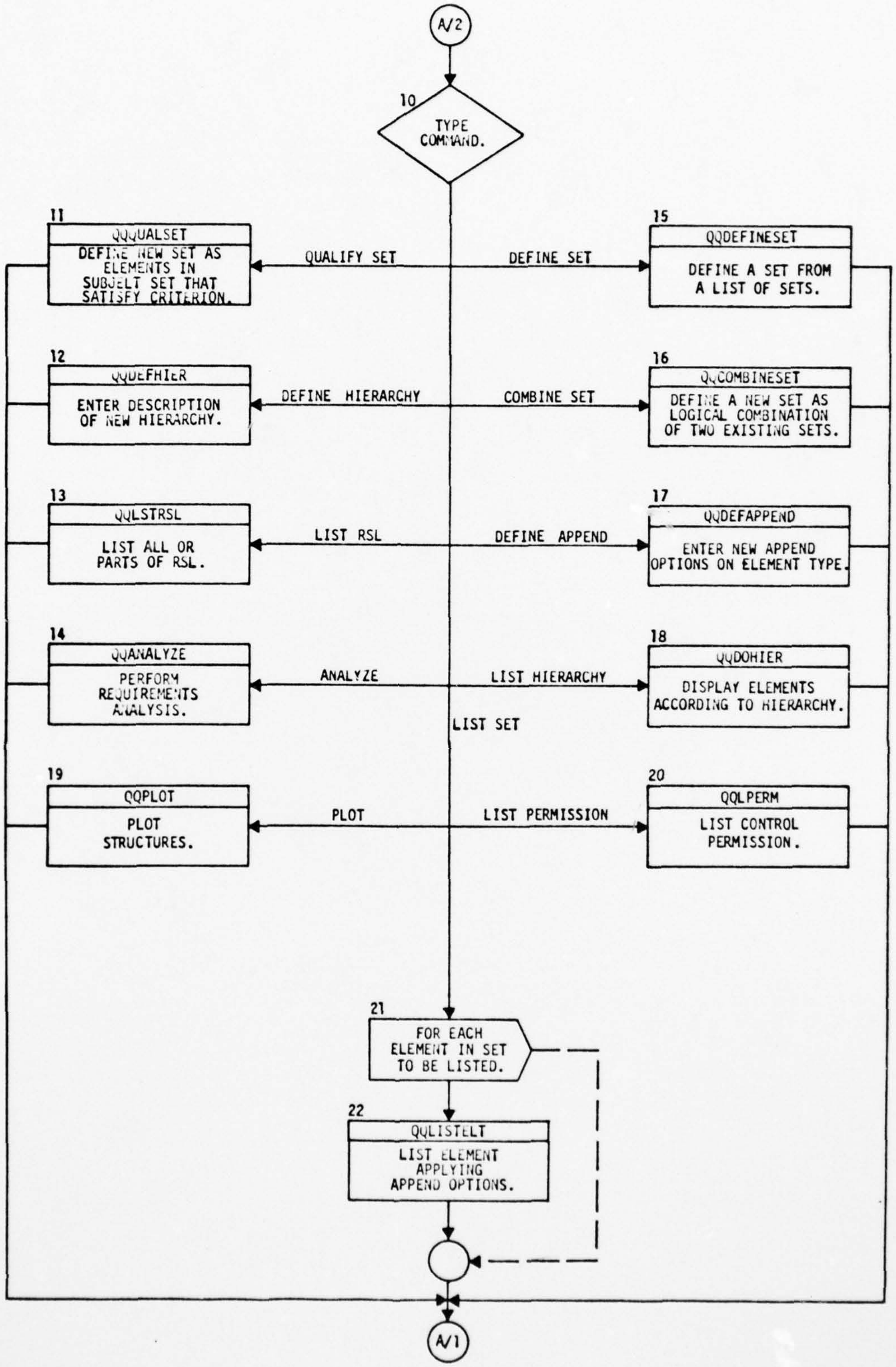



Figure 3-28 Requirements Analysis and Data Extraction (RADX) (Continued)

### 3.4.1 Define Set (QQDEFINESET)

Description

This module defines a new set as the union of a list of previously defined sets.

Input

| | | |
|---|---|---|
| MEMBER_LIST | - | List of sets used to define a set. |
| NEW_SET_NAME | - | The name of the new set that results from performing a DEFINE_SET, QUALIFY_SET, or COMBINE_SET operation. |
| COUNT_OPTION | - | Option to display number of members in newly defined set. |

Output

| | | |
|---|---|---|
| SET_DESCRIPTION_RECORD | - | Contains the name of the set, the location of the first APPEND_OPTION_RECORD, and the location of the first SET_MEMBER_RECORD. |
| SET_MEMBER_RECORD | - | Contains indexes into the ASSM_ELEMENT_ARRAY which identify the members of the set. |
| SET_LIST_ARRAY | - | An entry is made into this array for each set known to RADX. The entry contains the location of the SET_DESCRIPTION_RECORD. |

Processing

Figure 3-29 contains the functional flow diagram for this module.

Procedure References

The following correlates the functional processing elements in Figure 3-29 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | QQSTARTSET |
| [2-4] | - | QQSETDEF |
| [5-8] | - | QQENDSET |

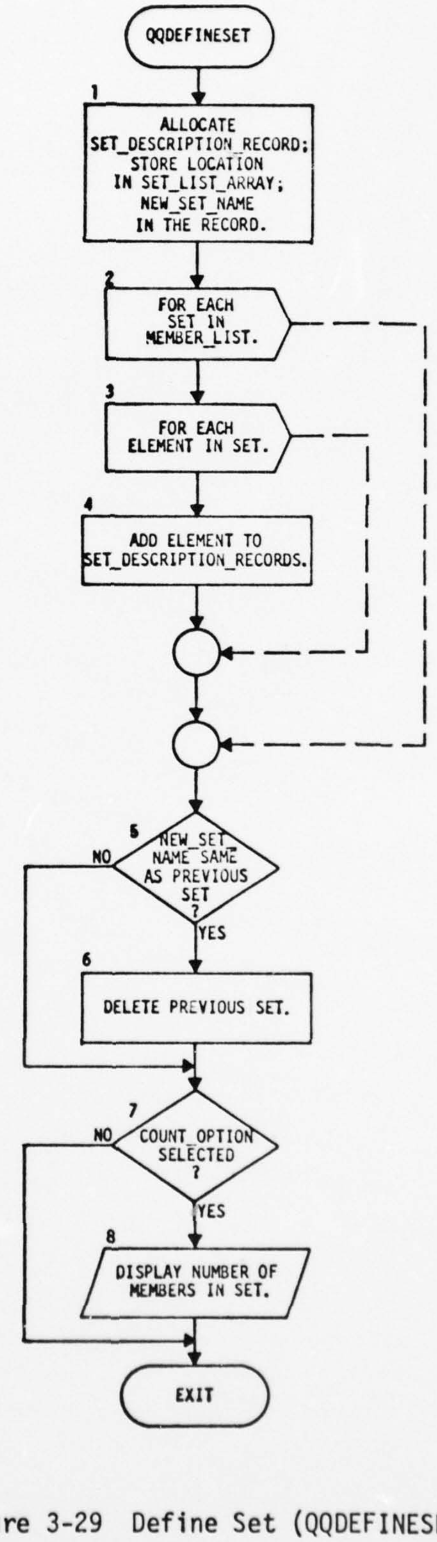
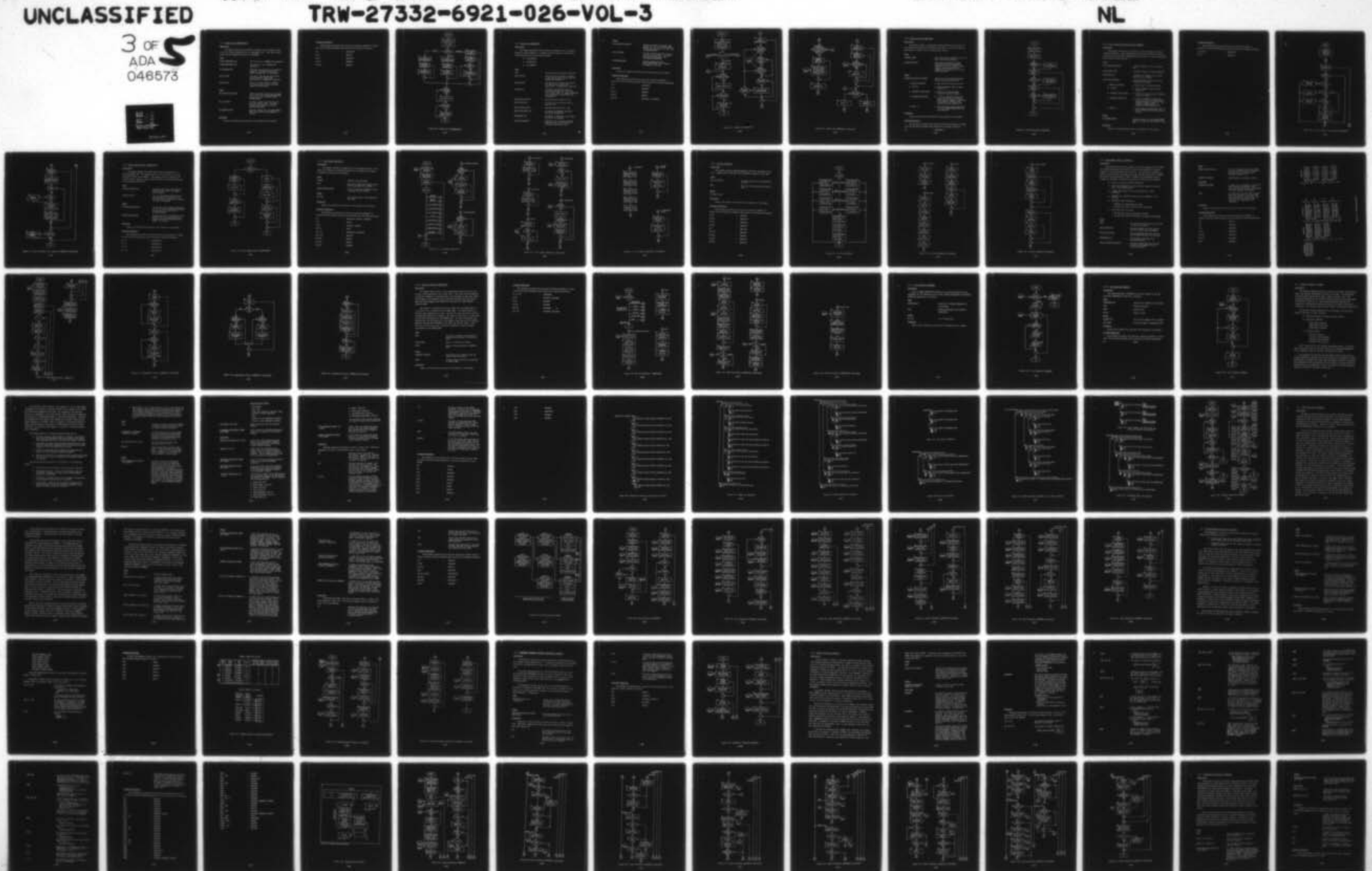



Figure 3-29 Define Set (QQDEFINESET)

AD-A046 573 TRW DEFENSE AND SPACE SYSTEMS GROUP HUNTSVILLE ALA F/G 9/2
REVS MAINTENANCE MANUAL. SREP FINAL REPORT. VOLUME III.(U)
AUG 77 W E BENOIT, P N BERGSTRESSER DASG60-75-C-0022
UNCLASSIFIED TRW-27332-6921-026-VOL-3 NL

3 OF 5
ADA
046573



3 OF 5

ADA
046573

1.0 2.8 2.5 3.15 2.2 5.0 5.6 3.5 2.0 6.3 1.1 4.0 7.1 4.5 1.8 1.25 1.4 1.6

NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

### 3.4.2 Combine Sets (QQCOMBINESET)

**Description**

This module derives the collection of members to be included in a new set as the logical combination of two independent sets. The logical combination can be INTERSECTION, UNION, or DIFFERENCE.

**Input**

| | | |
|---|---|---|
| FIRST_INDEPENDENT_SET | - | The first set of a COMBINE_SETS operation. |
| SECOND_INDEPENDENT_SET | - | The second set of a COMBINE_SETS operation. |
| TYPE_COMBINATION | - | Indicates how sets are to be logically combined. The combination can be UNION, INTERSECTION, or DIFFERENCE. |
| NEW_SET_NAME | - | The name of the new set that results from performing a DEFINE_SET, QUALIFY_SET, or COMBINE_SET operation. |
| COUNT_OPTION | - | Option to display number of members in a new set that results from the COMBINE_SETS operation. |

**Output**

| | | |
|---|---|---|
| SET_DESCRIPTION_RECORD | - | Contains the name of the set, the location of the first APPEND_OPTION_RECORD, and the location of the first SET_MEMBER_RECORD. |
| SET_LIST_ARRAY | - | An entry is made into this array for each SET known to RADX. The entry contains the location of the SET_DESCRIPTION_RECORD. |
| SET_MEMBER_RECORD | - | Contains indexes into the ASSM-ELEMENT_ARRAY which identify the members of the set. |

**Processing**

Figure 3-30 contains the functional flow diagram for this module.

Procedure References

The following correlates the functional processing elements in Figure 3-30 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | QQSTARTSET |
| [3-5] | - | QQORSET |
| [6-8] | - | QQDIFFSET |
| [9-12] | - | QQANDSET |
| [13-16] | - | QQENDSET |

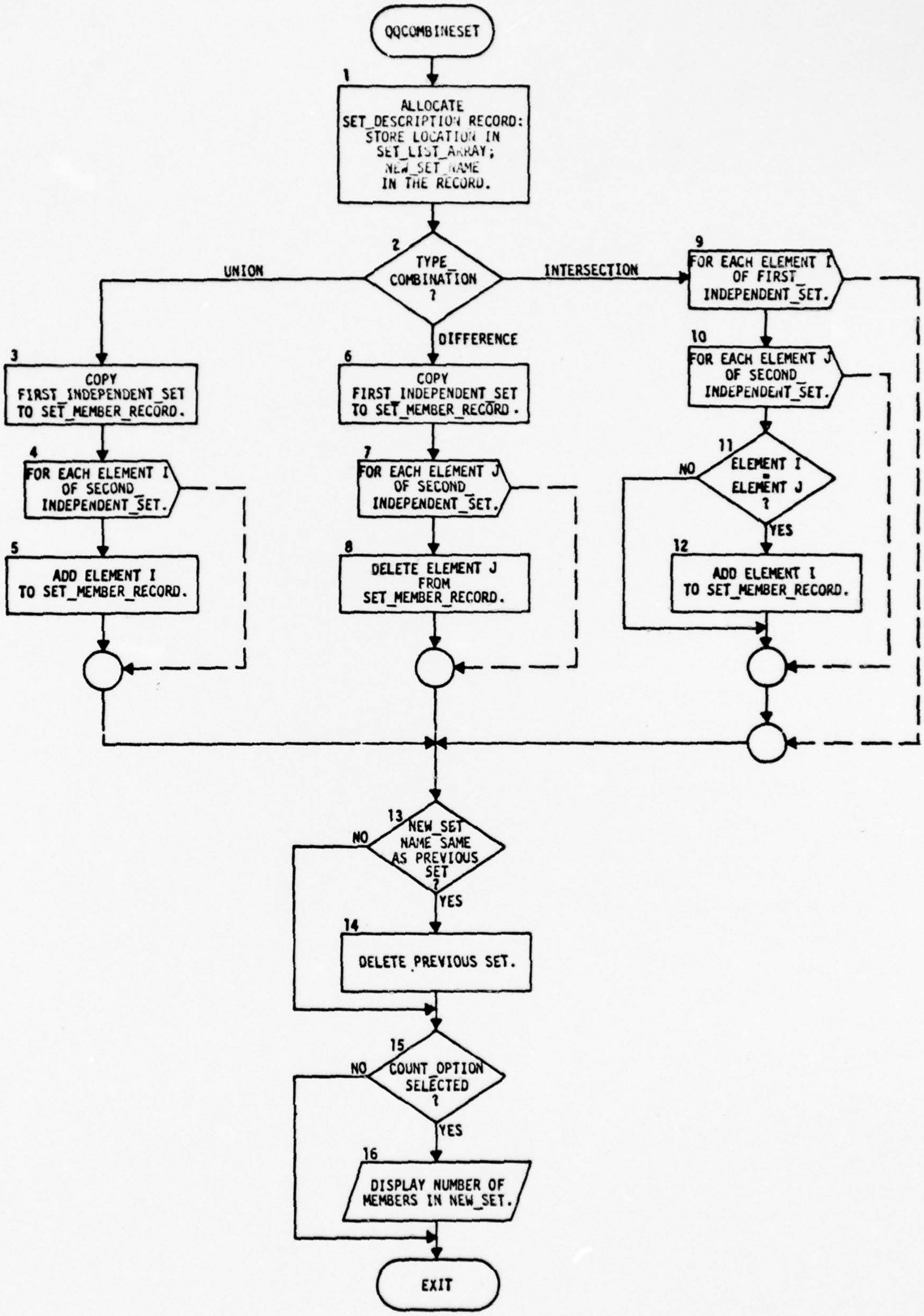



Figure 3-30 Combine Sets (QQCOMBINESET)

### 3.4.3 Qualify Set (QQQUALSET)

Description

This module determines the collection of elements to be included in NEW_SET as those elements in a subject set which satisfy a qualification criterion which can be one of the following:

- BY_ATTRIBUTE
- BY_RELATION
- BY_HIERARCHY.

Input

| | | |
|---|---|---|
| ASSM | - | Attribute and relationship instances. |
| COUNT_OPTION | - | Option to display number of members in the new set that results from the QUALIFY_SET operation. |
| NEW_SET_NAME | - | The name of the new set that results from performing a DEFINE_SET, QUALIFY_SET, or COMBINE_SET operation. |
| TYPE_QUALIFY | - | Indicates the technique used to qualify the members of an existing set to form a new set. Legal techniques are BY_ATTRIBUTE, BY_RELATION, and BY_HIERARCHY. |
| QUALIFYING_ATTRIBUTE | - | Attribute used to qualify a set. |
| QUALIFYING_VALUE | - | The value of an attribute used to qualify a set. |
| QUALIFYING_RELATION | - | Relation used to qualify a set. |
| QUALIFYING_OBJECT_SET | - | Collection of elements used when qualifying BY_RELATION. |
| INDEPENDENT_SET | - | Collection of elements to be listed, qualified, or analyzed. |
| SELECTED_HIERARCHY | - | Identifier of a previously defined hierarchy that is to be used for listing or qualifying a set. |

Output

| | | |
|---|---|---|
| SET_DESCRIPTION_RECORD | - | Contains the name of the set, the location of the first APPEND_OPTION_RECORD, and the location of the first SET_MEMBER_RECORD. |
| SET_LIST_ARRAY | - | An entry is made into this array for each set known to RADX. The entry contains the location of the SET_DESCRIPTION_RECORD. |
| SET_MEMBER_RECORD | - | Contains indexes into the ASSM_ELEMENT_ARRAY which identify the members of the set. |

Processing

Figure 3-31 contains the functional flow diagram for this module.

Procedure References

The following correlates the functional processing elements in Figure 3-31 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | QQSTARTSET |
| [3] | - | QQDOHIER |
| [4, 5] | - | QQENDSET |
| [8-14] | - | QQBYATT |
| [15-25] | - | QQBYRSLREL, QQBYIMPREL |

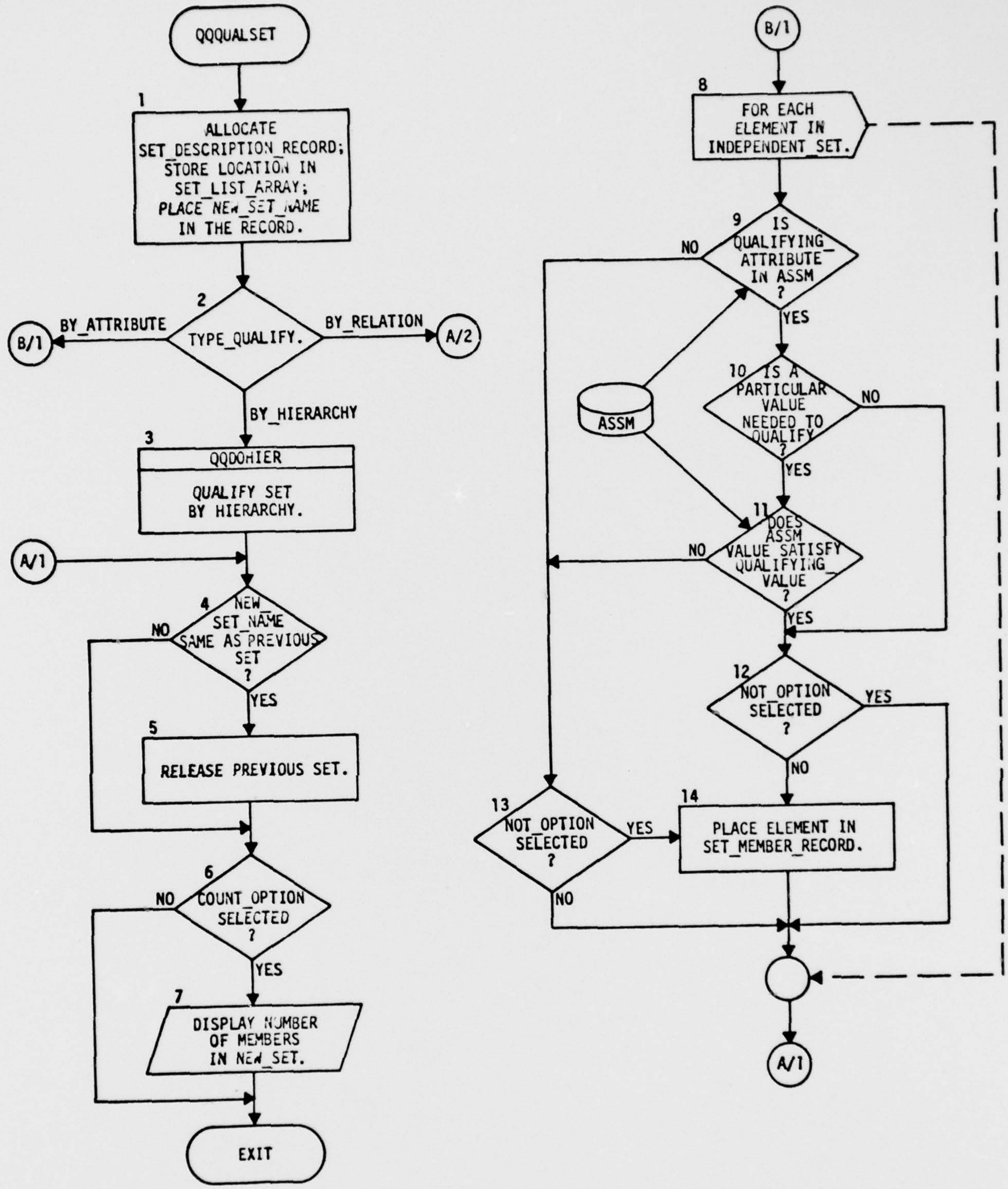



Figure 3-31 Qualify Set (QQQUALSET)

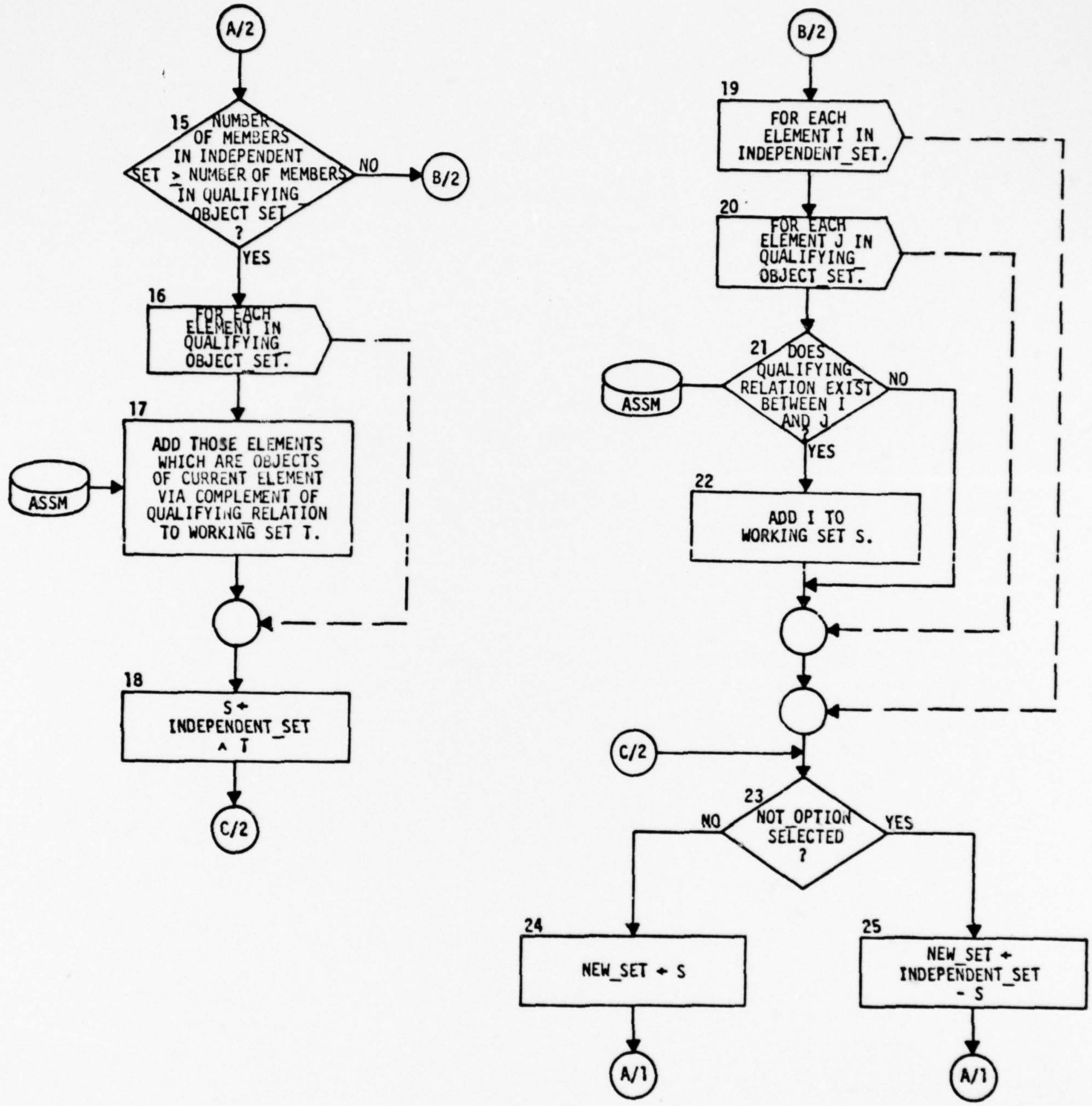



Figure 3-31 Qualify Set (QQQUALSET) (Continued)

### 3.4.4 Define Hierarchy (QQDEFHIER)

Description

This module accepts a translated hierarchy definition and stores it with hierarchy connectivity information such that it can be used later for qualifying or listing a set in a hierarchical manner.

Input

| | | |
|---|---|---|
| HIERARCHY_NAME | - | Name used to apply hierarchy in a list set or qualify set command. |
| HIERARCHY_ENTRIES | - | Each entry contains the triplet (SUBJECT_TYPE, BINDING_RELATION, OBJECT_TYPE) which is used to trace elements through the ASSM from SUBJECT_TYPE to OBJECT_TYPE via the BINDING_RELATION. |

Output

| | | |
|---|---|---|
| HIERARCHY_DESCRIPTION_RECORD | - | Storage of a user defined hierarchy which contains the following: |

| | |
|---|---|
| • NAME_OF_HIERARCHY | • Name given to hierarchy by user. |
| • TOP_TYPE | • Type of elements that can begin hierarchy. |
| • HIERARCHY_START_POINTS | • Entries in hierarchy where hierarchy tracing can begin. |
| • HIERARCHY_OPERATIONS | • A list of ordered triplets that direct the tracing of relationships between elements in the ASSM. The triplet is (SUBJECT_TYPE, BINDING_RELATION, OBJECT_TYPE). |
| • FROM, TO | • Two arrays that specify the order in which HIERARCHY_OPERATIONS are to be applied. |

Processing

Figure 3-32 contains the functional flow diagram for this module.

Procedure Reference

The following correlates the functional processing elements in Figure 3-32 with the REVS procedures which perform the indicated processing.

[3] - QQHIERMATCH

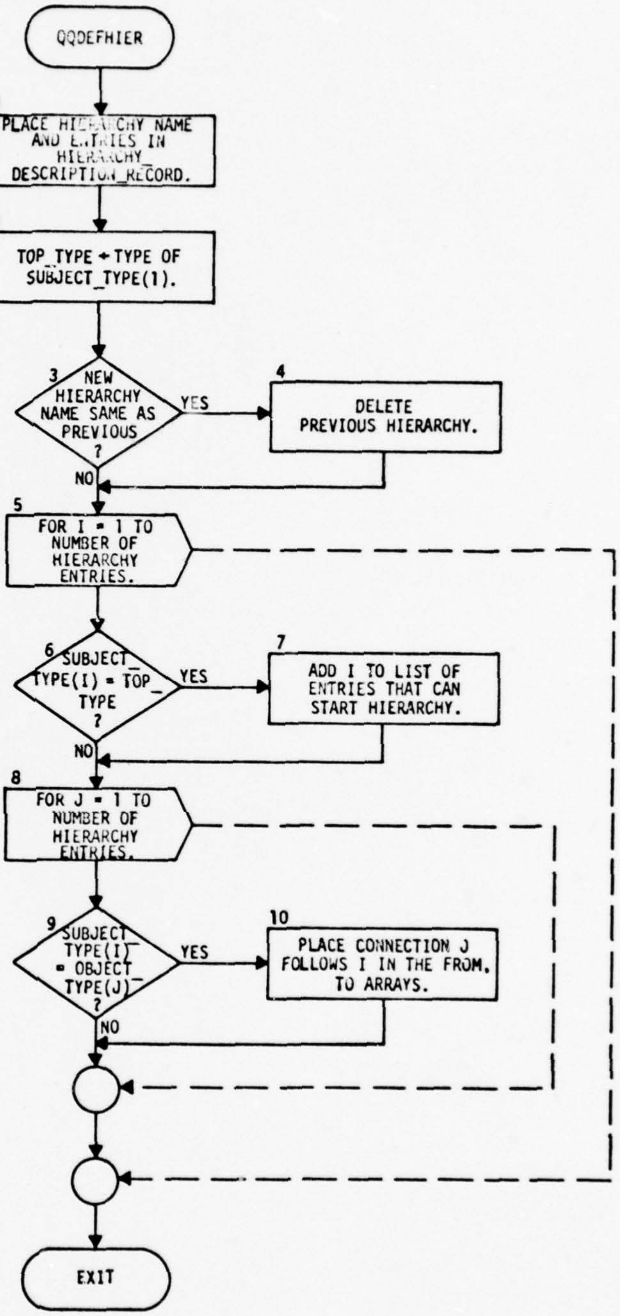



Figure 3-32 Define Hierarchy (QQDEFHIER)

### 3.4.5 List or Qualify Set by Hierarchy (QQDOHIER)

Description

This module either lists or qualifies a set of elements according to a defined hierarchy. Those elements which are listed or placed in the new set must be in the original independent set and they must also be encountered while traversing the hierarchy.

Input

| | | |
|---|---|---|
| LIST_OR_QUALIFY_OPTION | - | Indicates whether to list or qualify set. |
| HIER_DISPLAY_FORM | - | Indicates format for listing hierarchy. |
| INDEPENDENT_SET | - | Collection of elements to be listed, qualified, or analyzed. |
| QUALIFYING_HIERARCHY | - | Hierarchy used to qualify a set containing the following: |
| • NAME_OF_HIERARCHY | | • Name given to hierarchy by user. |
| • TOP_TYPE | | o Type of elements that can begin hierarchy. |
| • HIERARCHY_START_POINTS | | • Entries in hierarchy where hierarchy tracing can begin. |
| • HIERARCHY_OPERATIONS | | • A list of ordered triplets that direct the tracing of relationships between elements in the ASSM. The triplet is (SUBJECT_TYPE, BINDING_RELATION, OBJECT_TYPE). |
| • FROM, TO | | • Two arrays that specify the order in which HIERARCHY_OPERATIONS are to be applied. |

Output

| | | |
|---|---|---|
| SET_MEMBER_RECORD | - | Contains indexes into the ASSM_ELEMENT_ARRAY which identify the members of the set. |

Processing

Figure 3-33 contains the functional flow diagram for this module.

Procedure References

The following correlates the functional processing elements in Figure 3-33 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [2, 12-18] | - | QQCONTHIER |
| [4-11] | - | QQINCLUDE |

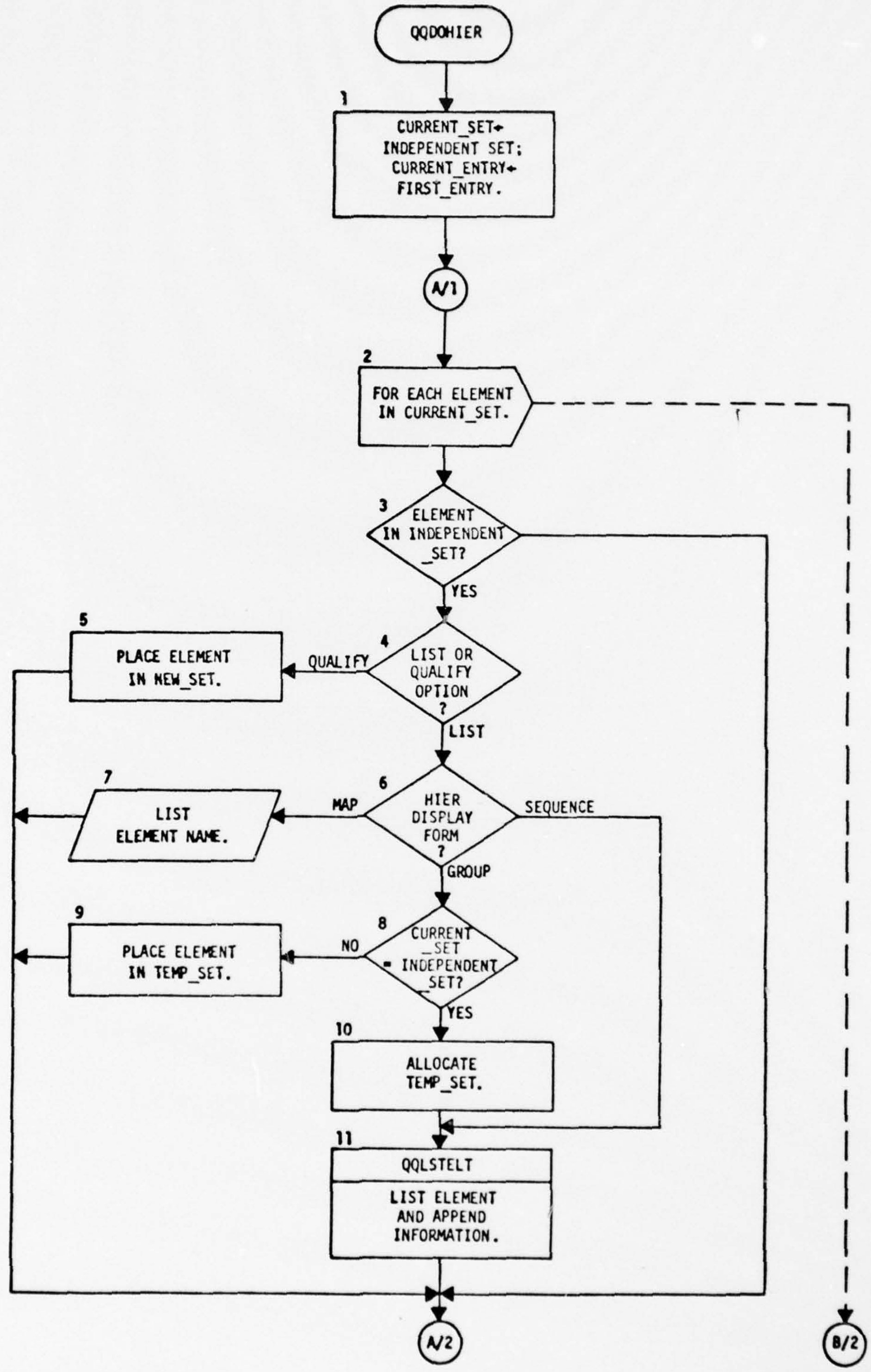



Figure 3-33 List or Qualify Set by Hierarchy (QQDOHIER)

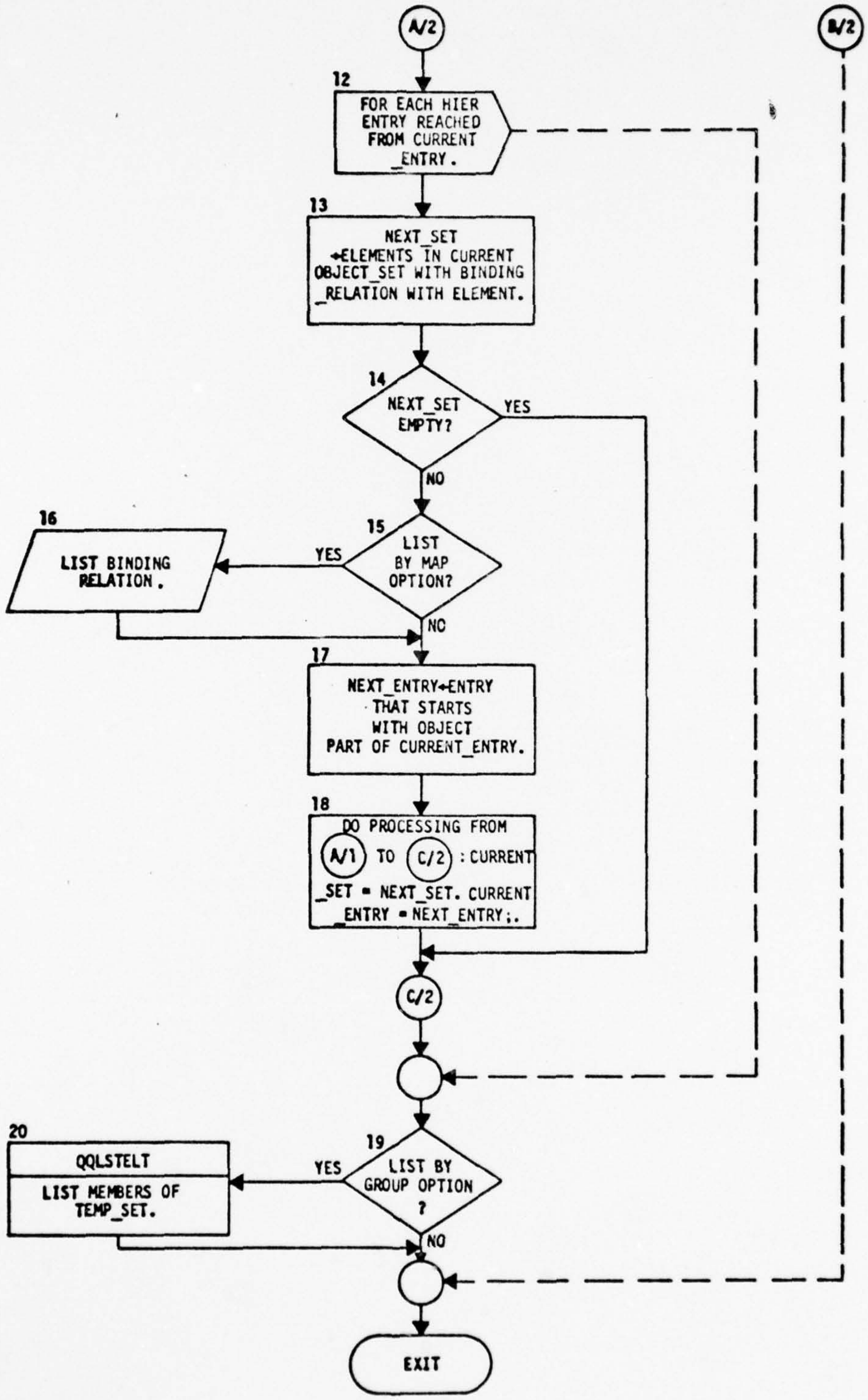



Figure 3-33 List or Qualify Set by Hierarchy (QQDOHIER) (Continued)

### 3.4.6 Define Append Options (QQDEFAPPEND)

Description

This module updates the append options that are attached to a SELECTED_APPEND_TYPE. The append options for all TYPE_SETs or for one selected TYPE_SET can be updated. The update consists of deleting the current options, making a copy of the new options, and attaching the copy to the SELECTED_APPEND_TYPE.

Input

| | | |
|---|---|---|
| SELECTED_APPEND_TYPE | - | Indicates the element type that has been selected to have its APPEND_OPTION changed. |
| APPEND_ITEM_LIST | - | List of associated information such as relations and attributes to be displayed when an element of a particular type is displayed. |

Output

| | | |
|---|---|---|
| SET_DESCRIPTION_RECORD | - | Contains the name of the set, the location of the first APPEND_OPTION_RECORD, and the location of the first SET_MEMBER_RECORD. |
| APPEND_OPTION_RECORD | - | Contains one type of information to be displayed when an element that is a subset of the owning SET_DESCRIPTION_RECORD is displayed. |

Processing

Figure 3-34 contains the functional flow diagram for this module.

Procedure References

The following correlates the functional processing elements in Figure 3-34 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [2, 3] | - | QQDISPAPLIST |
| [4, 5] | - | QQCOPYAPLIST |
| [7, 8] | - | QQDISPAPLIST |
| [9, 10] | - | QQCOPYAPLIST |

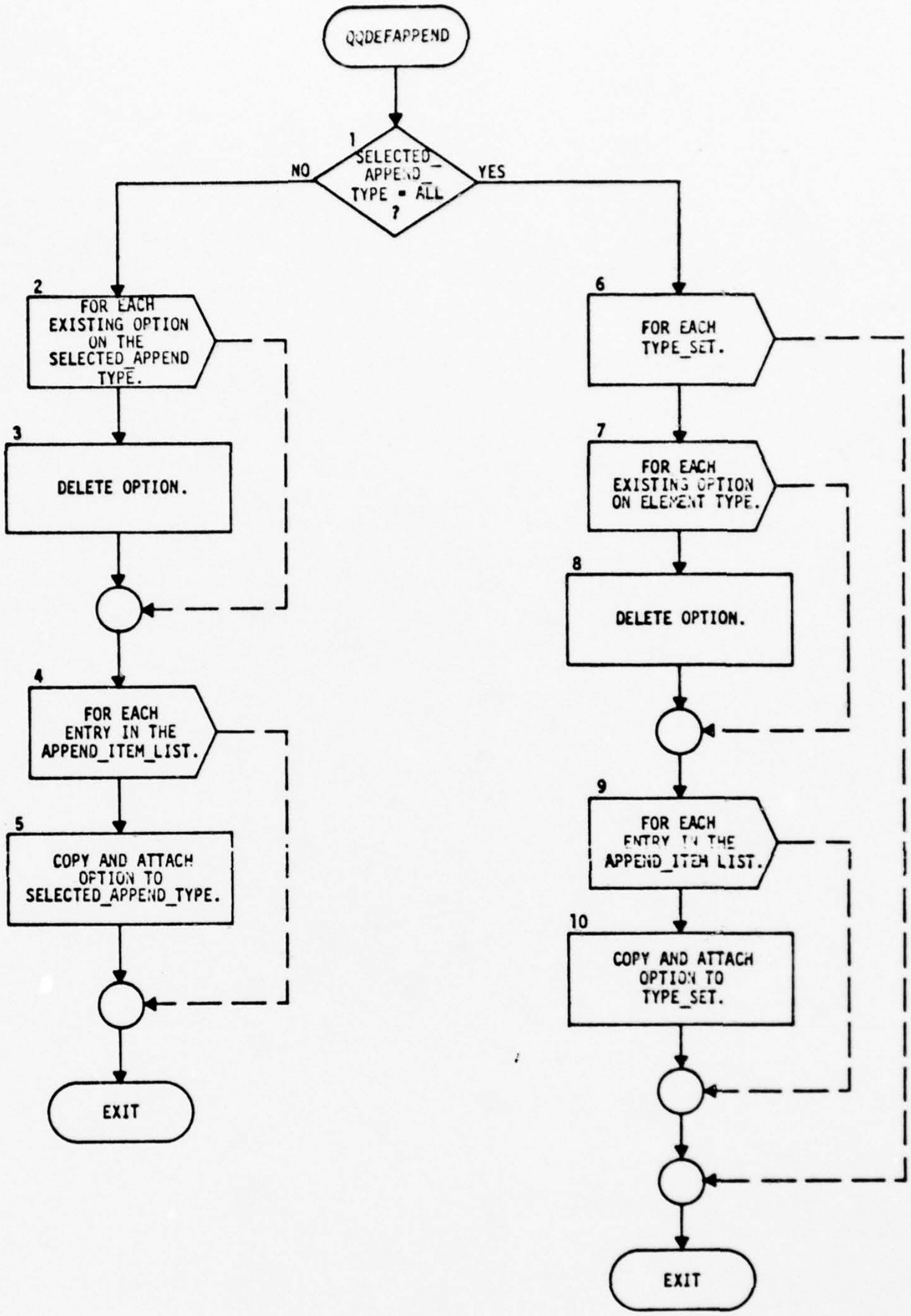



Figure 3-34 Define Append Options (QQDEFAPPEND)

### 3.4.7 List Element (QQLISTELT)

Description

This module displays an element and its associated relations, attributes, and structural information according to the append option that is in effect for the type of the element.

Input

| | | |
|---|---|---|
| ELEMENT | - | Element to be displayed. |
| ASSM | - | Any part of ASSM that contains information associated with element. |
| APPEND_OPTION_RECORD | - | List of associated information to be displayed with the element. |

Output

| | | |
|---|---|---|
| RSL TEXT | - | All displays made by this module are legal RSL. |

Processing

Figure 3-35 contains the functional flow diagram for this module.

Procedure References

The following correlates the functional processing elements in Figure 3-35 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | QQENDLINE, QQPUTELT, QQPUTCMT |
| [2] | - | QQRETS |
| [5-8] | - | QQLAPREL, QQLACREL |
| [9-11] | - | QQLAT |
| [12, 13] | - | QQLALLPREL, QQLALLCREL |
| [14, 15] | - | QQLALLAT |
| [16-18] | - | QQLSTR |
| [19-22] | - | QQLREFS |
| [23-27] | - | QQLREFBY |

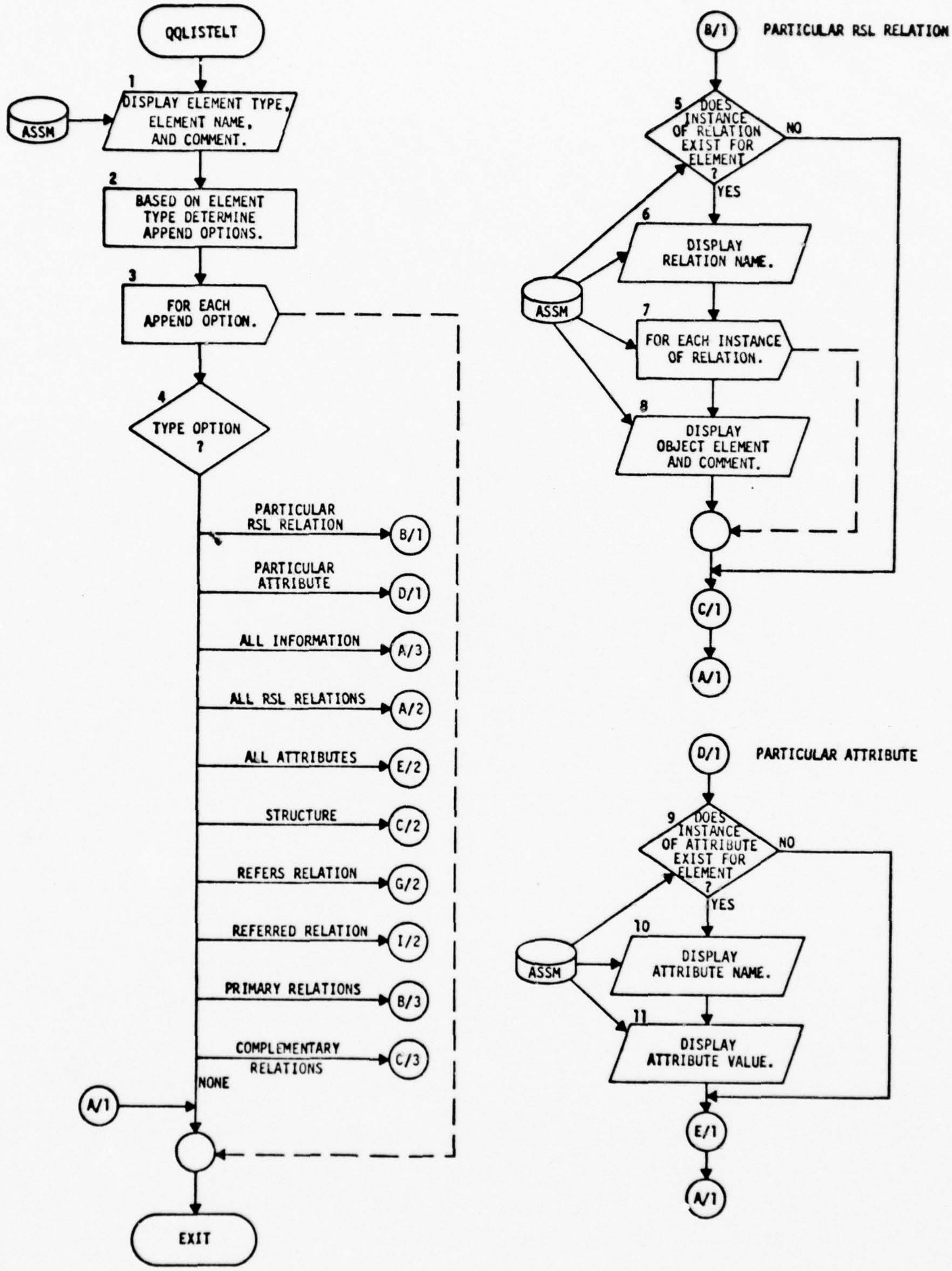



Figure 3-35 List Element (QQLISTELT)

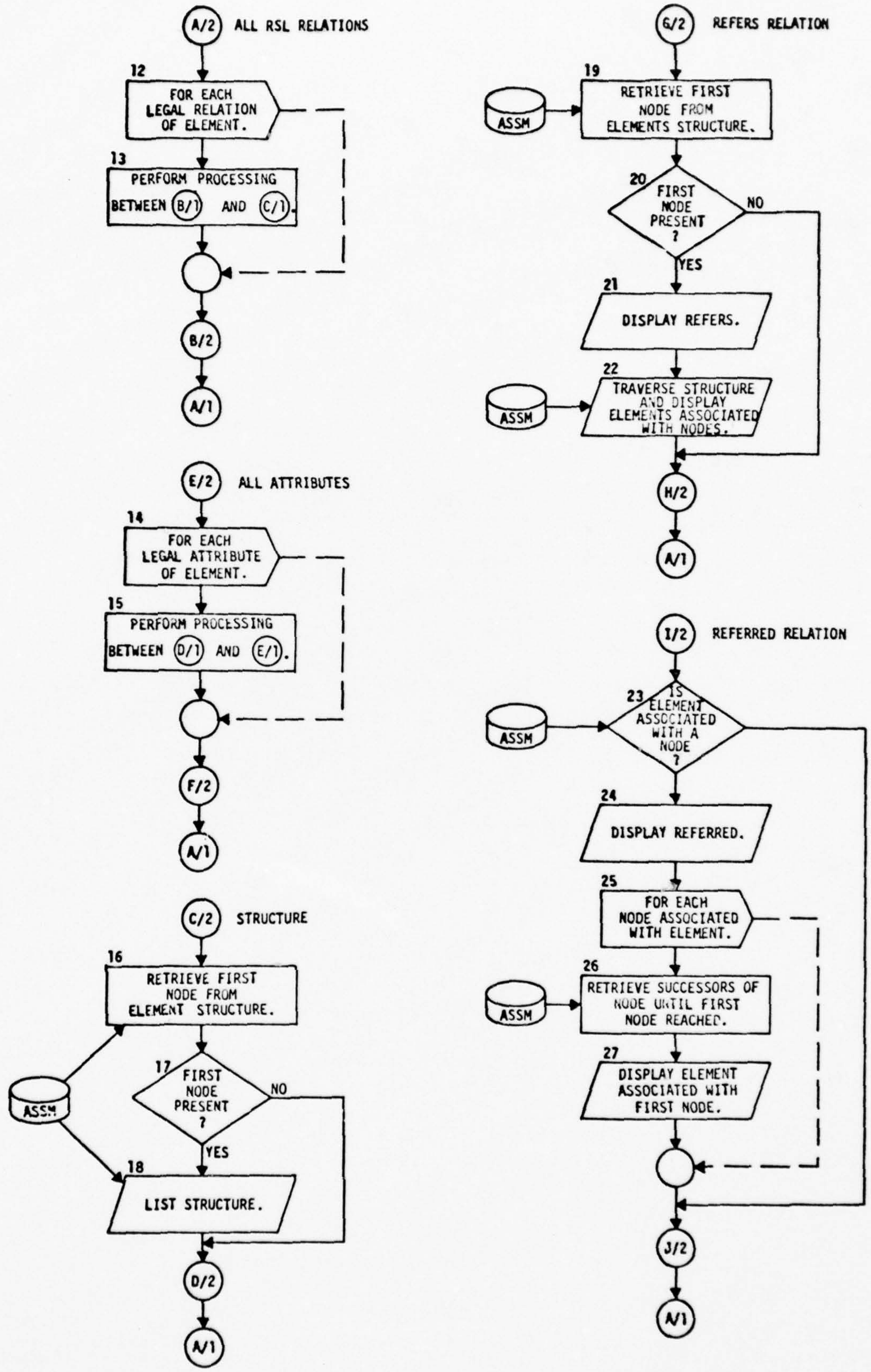



Figure 3-35 List Element (QQLISTELT) (Continued)

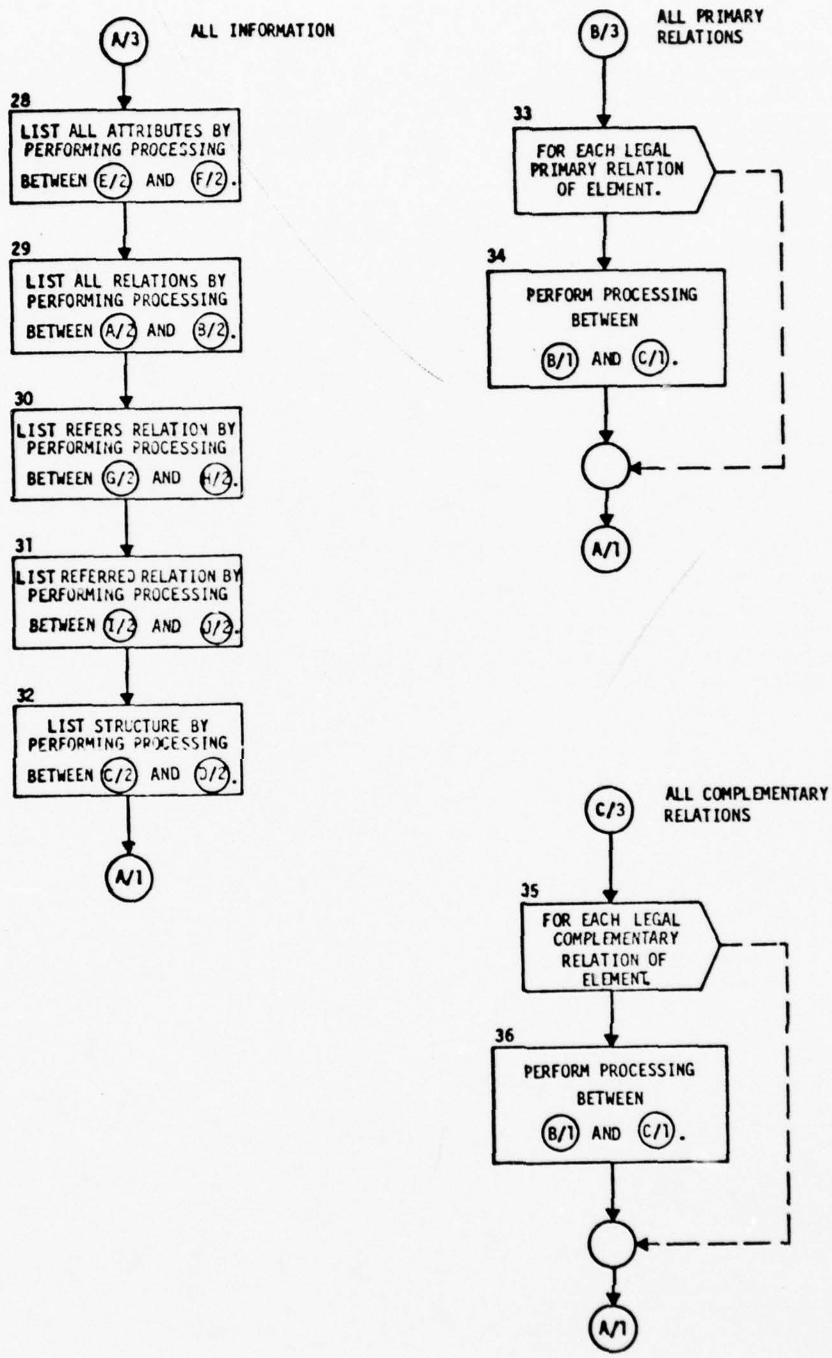



Figure 3-35 List Element (QQLISTELT) (Continued)

### 3.4.8 List RSL (QQLSTRSL)

Description

This module displays the descriptions of the basic components of RSL (i.e., ELEMENT_TYPEs, RELATIONs, and ATTRIBUTEs) contained in the ASSM.

Input

| | | |
|---|---|---|
| RSL_LIST_OPTION | - | Indicates portion of RSL description to list. |
| ASSM | - | That part which contains RSL description. |

Output

RSL Description

Processing

Figure 3-36 contains the functional flow diagram for this module.

Procedure References

The following correlates the functional processing elements in Figure 3-36 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1-5] | - | QQLSTRSL |
| [6,10] | - | QQRSLAETP |
| [7,11] | - | QQRSLAREL |
| [8,12] | - | QQRSLAATT |
| [9] | - | QQRSLSUM |
| [13-15] | - | QQRSLTYPE |
| [16-23] | - | QQRSLREL |
| [24-29] | - | QQRSLATT |
| [30-38] | - | QQRSLSELT |

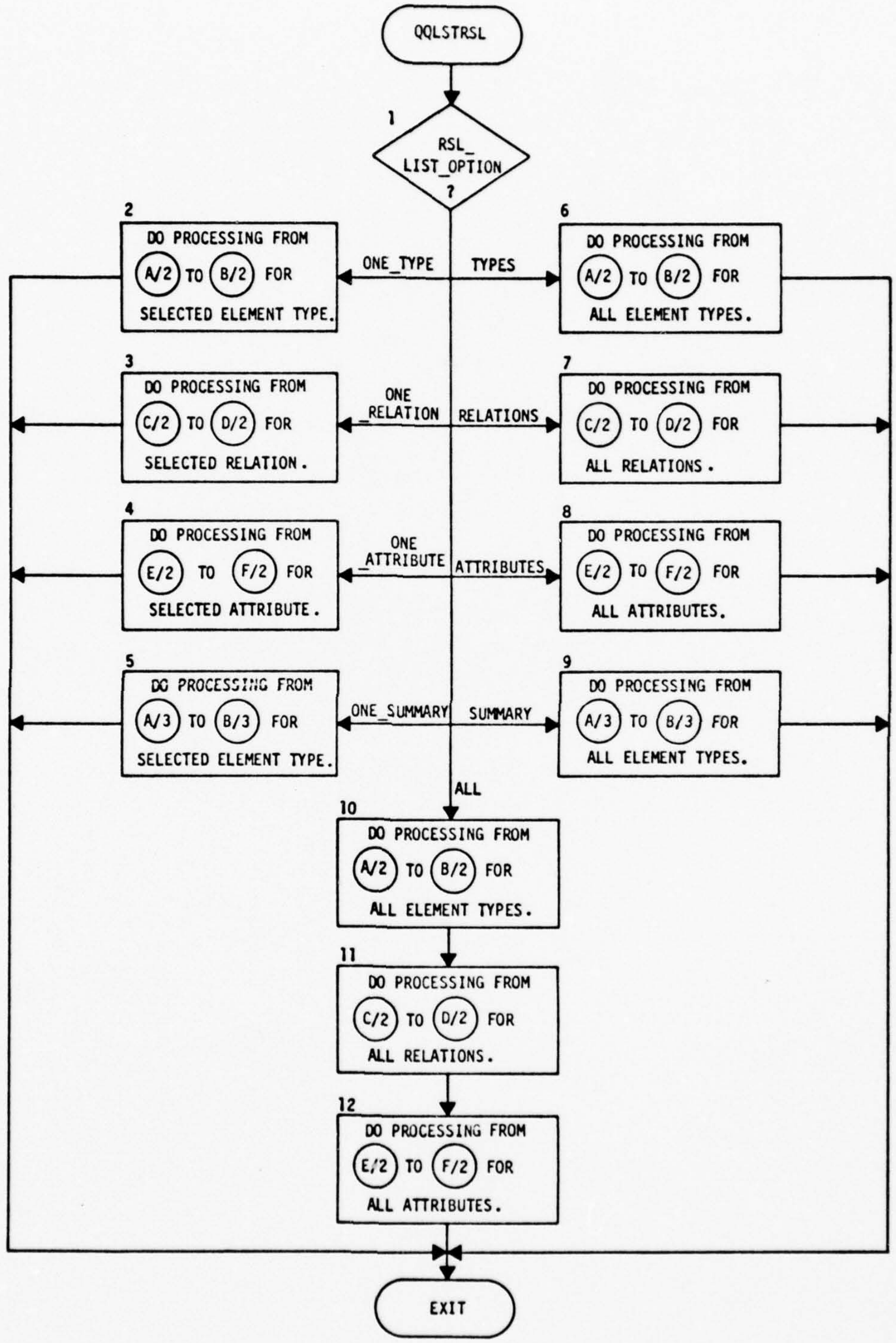



Figure 3-36 List RSL (QQLSTRSL)

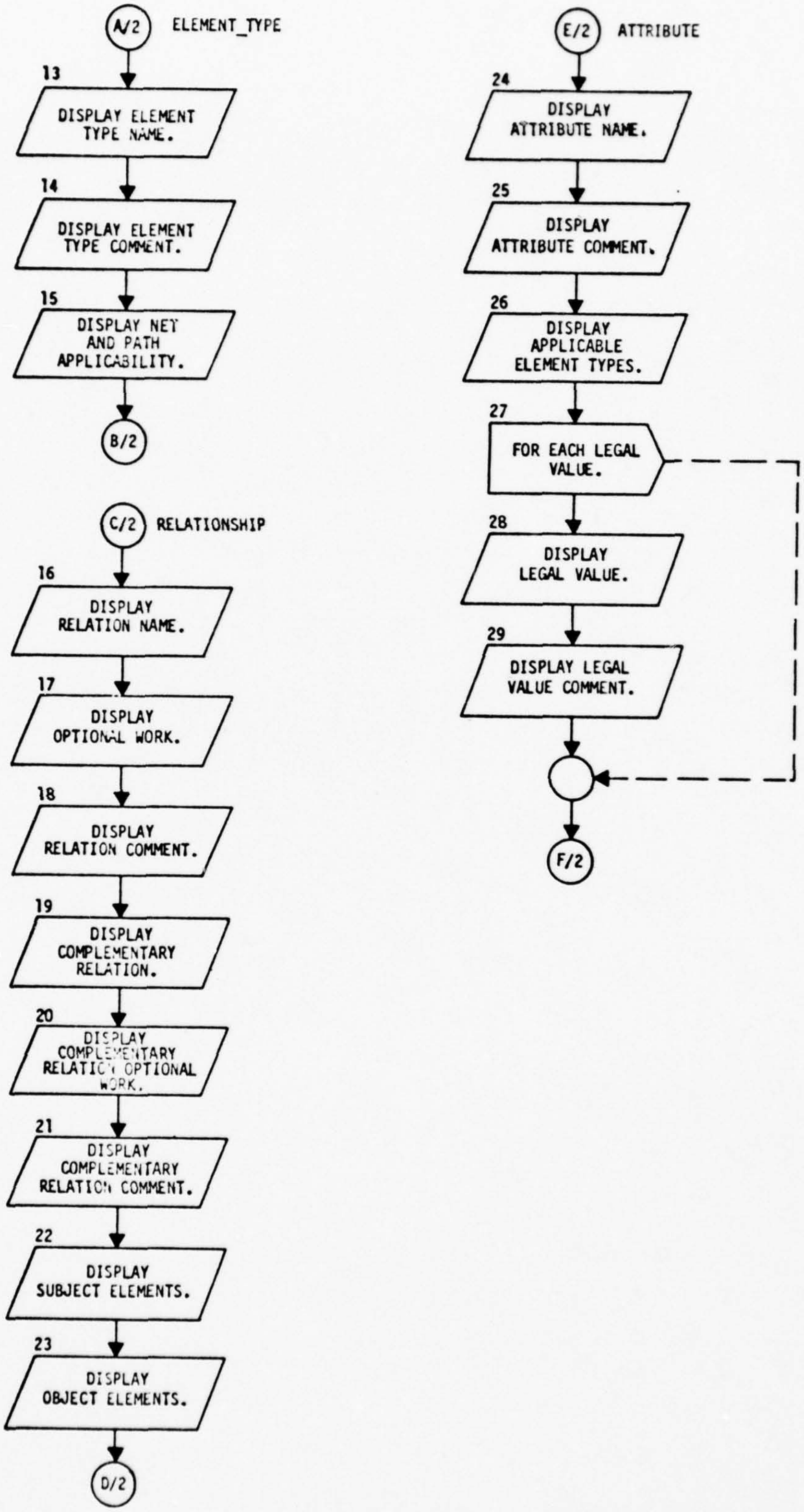



Figure 3-36 List RSL (QQLSTRSL) (Continued)

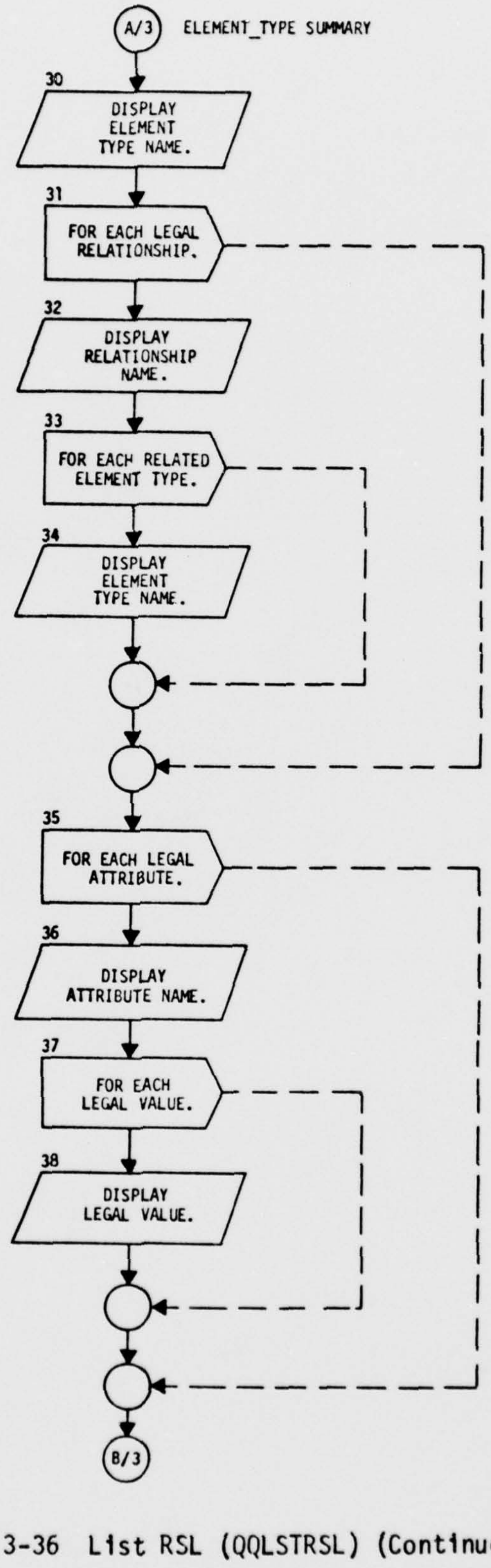



Figure 3-36 List RSL (QQLSTRSL) (Continued)

### 3.4.9 Requirements Analysis (QQANALYZE)

Description

This module serves the dual purpose of selecting elements from the ASSM for simulation/analysis and statically analyzing the elements to identify anomalies in the requirements specifications. The selection of elements is based on a set of R-Nets provided by the user (through either RADX or SIMGEN), the type of simulation/analysis (BETA or GAMMA) to be performed, and the contents of the ASSM. The analysis consists of testing for the following:

- Loops in a data or structure hierarchy.
- Data having membership in more than one repetitive data set (e.g., FILE, ENTITY_CLASS).
- Illegal specification of the USE attribute in a data hierarchy.
- LOCALITY of a repetitive data set and its members is not the same.
- Sequential data flow errors
  - Reference to unassigned data values
  - Assigned data values that are never referenced.
- Concurrent data flow errors
  - The same data values concurrently assigned
  - The same data values concurrently assigned and referenced.

Input

| | | |
|---|---|---|
| ASSM | - | That portion of ASSM pertaining to flow-oriented requirements. |
| DATA_FLOW_OPTION | - | Indicates whether data flow analysis should or should not be performed. |
| TYPE_OF_ACTIVATION | - | Indicates whether RADX was activated by the SIMGEN function or by the user. |
| INDEPENDENT_SET | - | The collection of R-Nets to be analyzed/simulated. |
| TYPE_OF_SIMULATION/ANALYSIS | - | Indicates whether data with USE BETA or data with USE GAMMA is to be analyzed/simulated. |

Output

| | | |
|---|---|---|
| SIMGEN TRANSLATION LIST | - | A list of elements used by the SIMGEN function to determine what elements to translate and consolidate into a simulator (Section 3.5). |
| DIAGNOSTIC MESSAGES | - | Identification of structural errors. |

Local Data

| | | |
|---|---|---|
| INFORMATION NETWORK | - | A linked list illustrated in Figure 3-37 which contains the elements to be analyzed/simulated and the relationships that exist between the elements. |
| TOKEN | - | An information container that is moved along the structure of an R-Net when data flow analysis is performed. The TOKEN provides the status of data accessible to a node when it is traversed. |

Processing

Figure 3-38 contains the functional flow diagram for this module.

Procedure References

The following correlates the functional processing elements in Figure 3-38 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | QQZINIT |
| [2] | - | QQINFOGEN |
| [3] | - | QQZSUBSET |
| [4-14] | - | QQMEMTEST |
| [15-17] | - | QQZATTSET |
| [18-33] | - | QQLOCTEST |
| [40] | - | QQSIMINIT |

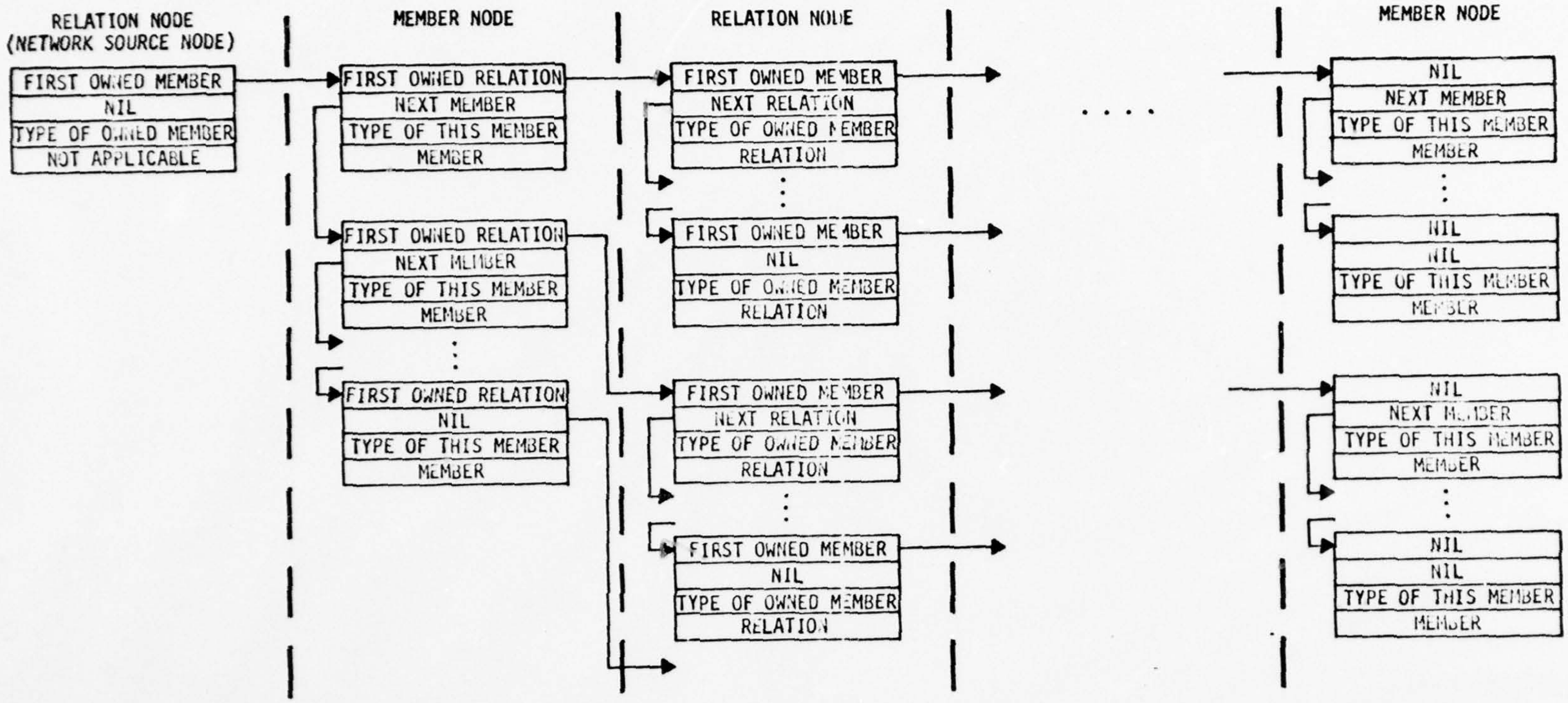



Figure 3-37 Information Network

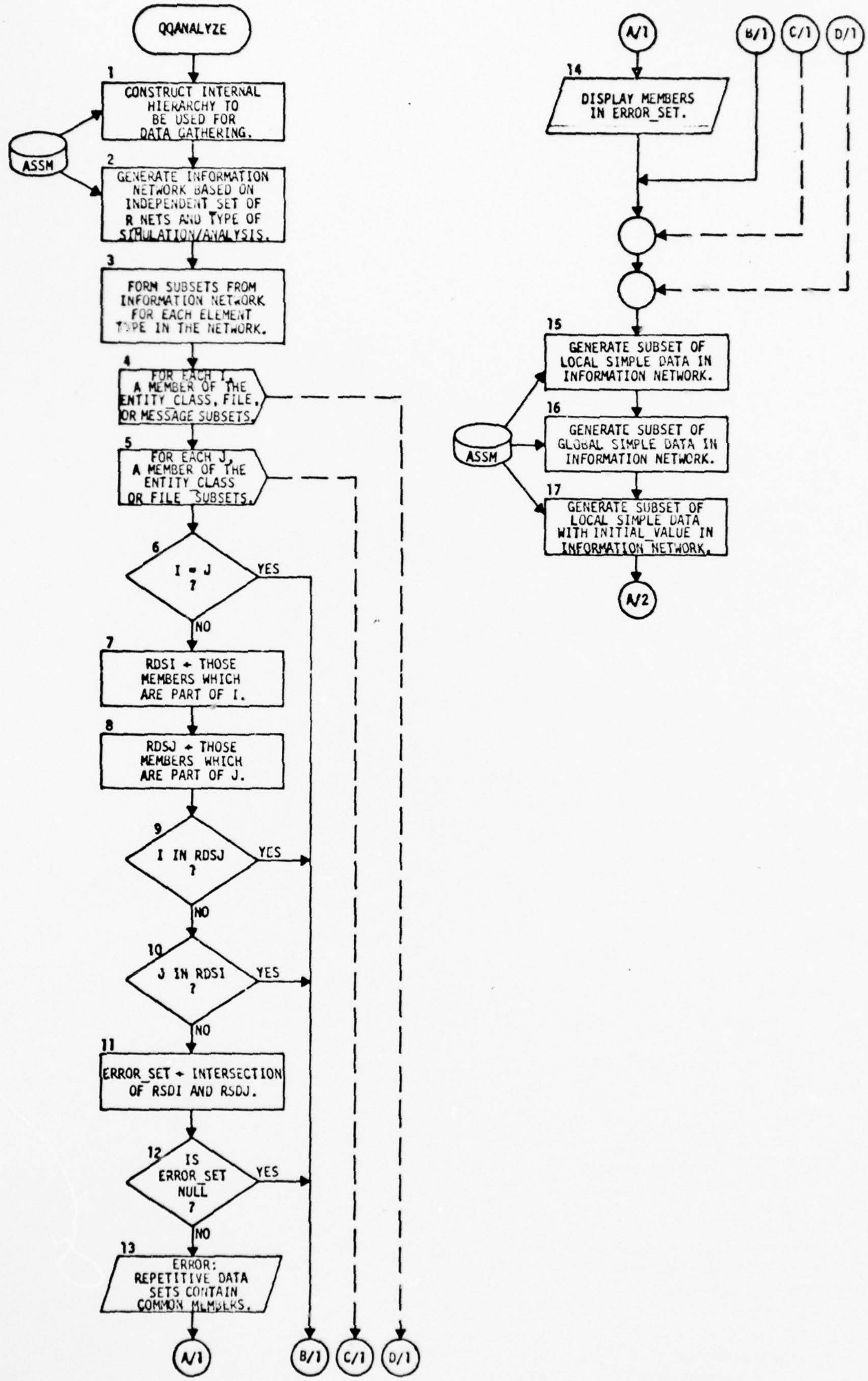



Figure 3-38 Requirements Analysis (QQANALYZE)

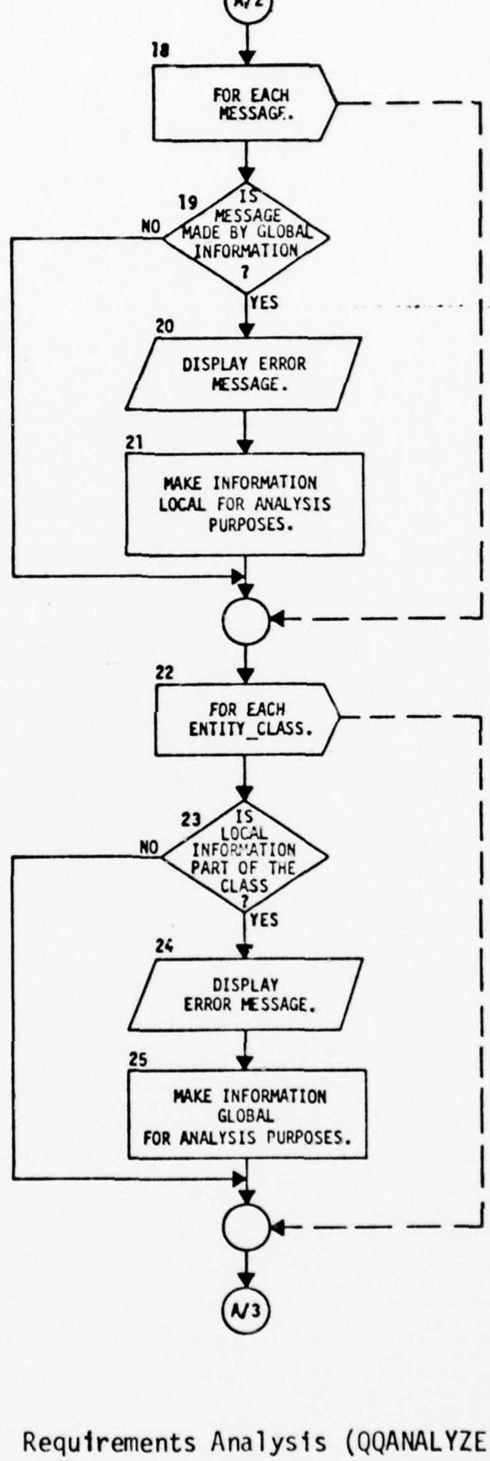



Figure 3-38 Requirements Analysis (QQANALYZE) (Continued)

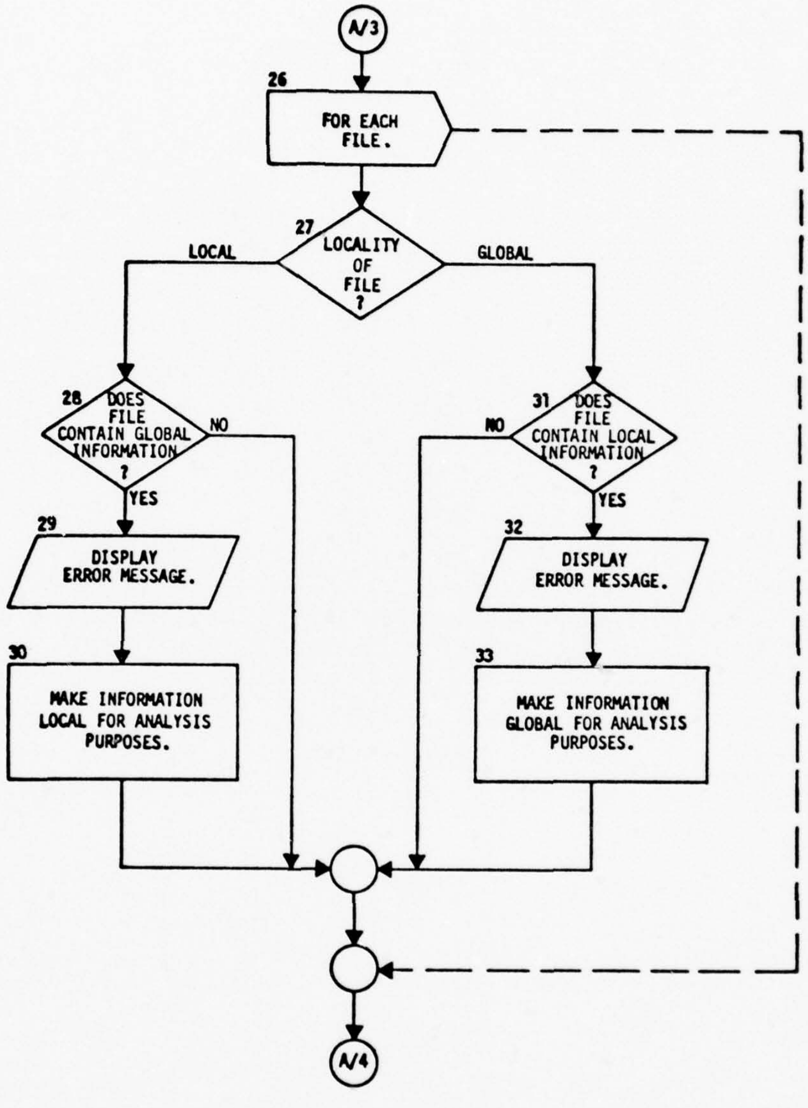



Figure 3-38 Requirements Analysis (QQANALYZE) (Continued)

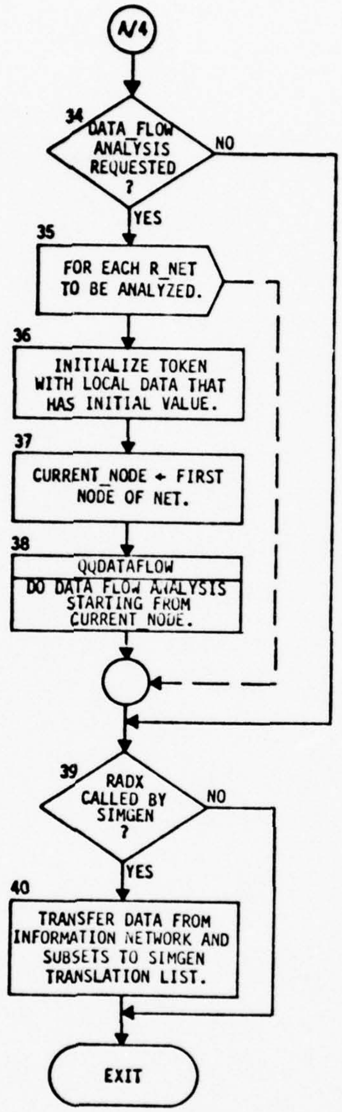



Figure 3-38 Requirements Analysis (QQANALYZE) (Continued)

### 3.4.10 Data Flow Analysis (QQDATAFLOW)

Description

This module detects errors in the requirements specifications by performing a data flow analysis for a given R-Net. The errors that are detected include the reference to unassigned data, the assignment of data that cannot be referenced, the assignment of the same data from more than one parallel path, and the assignment and reference of the same data from different parallel paths.

The analysis is performed by moving a TOKEN from the CURRENT_NODE in a structure to all successors of the CURRENT_NODE. The TOKEN contains the status of data accessible by a node. The test for data flow errors consists of determining whether the contents of the TOKEN is consistent or inconsistent with the data requirements of the CURRENT_NODE. The data requirements are obtained from the relationships of the elements associated with the CURRENT_NODE and from the data referenced by OR nodes and FOR EACH nodes. After the test for data flow errors, the TOKEN is updated to reflect the data requirements of the CURRENT_NODE.

Input

| | | |
|---|---|---|
| ASSM | - | Structures, elements associated with nodes and node branches, and relationships. |
| CURRENT_NODE | - | Node in a structure to analyze. |
| TOKEN | - | Status of data accessible to CURRENT_NODE. |

Output

| | | |
|---|---|---|
| DIAGNOSTIC MESSAGES | - | Identification of sequential and concurrent data flow errors. |
| TOKEN | - | Status of data accessible to successors of CURRENT_NODE. |

Processing

Figure 3-39 contains the functional flow diagram for this module.

Procedure References

The following correlates the functional processing elements in Figure 3-39 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1, 10-12] | - | QQTRAVERSE |
| [2-6] | - | QQFLWALVP, QQFLWEVT |
| [7-9] | - | QQFLWSNT |
| [13-27] | - | QQFLOWOR |
| [28-37] | - | QQFLOWAND |
| [38, 42] | - | QQFLOWFOR, QQFLOWSLT |

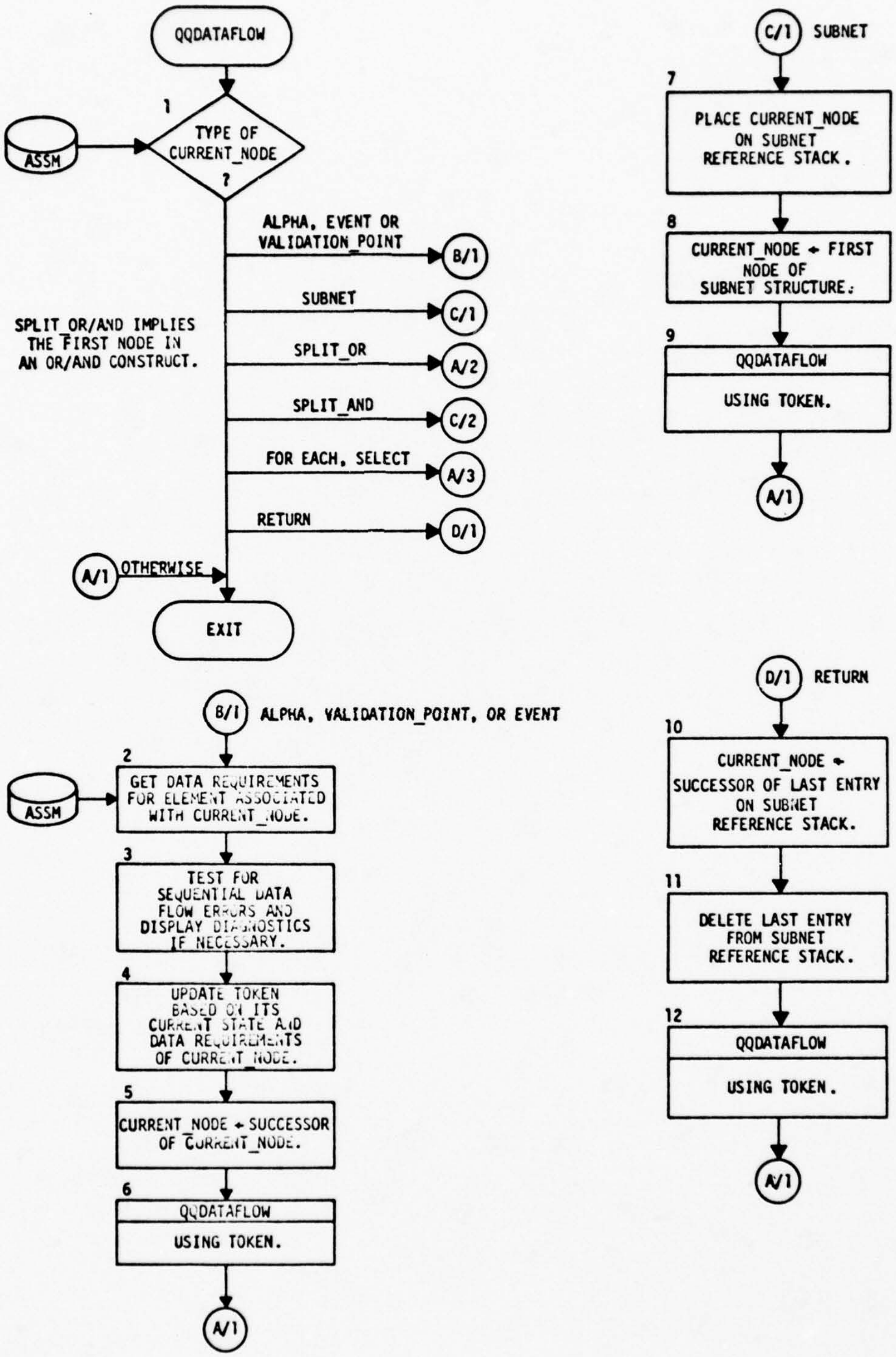



Figure 3-39 Data Flow Analysis (QQDATAFLOW)

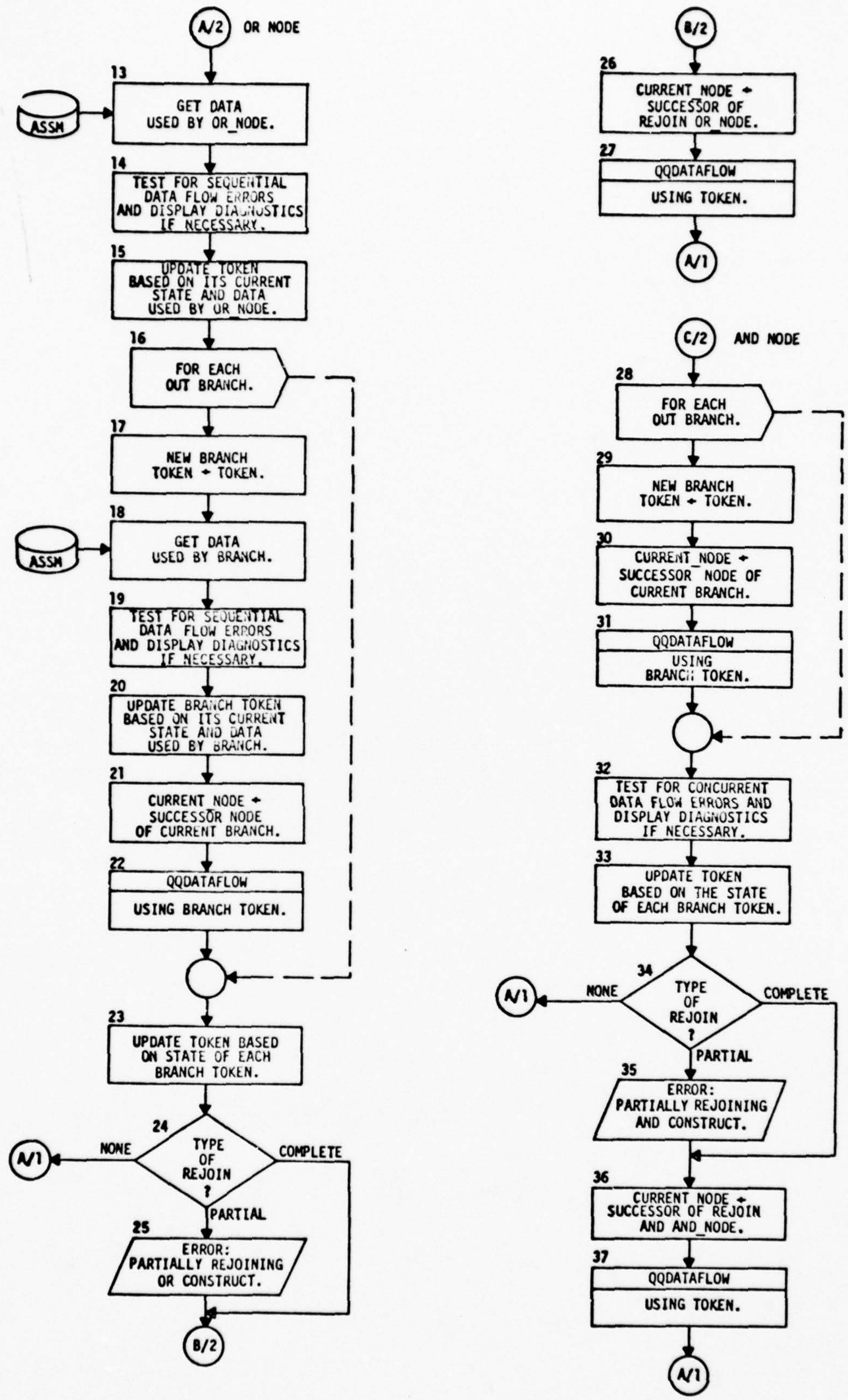



Figure 3-39 Data Flow Analysis (QQDATAFLOW) (Continued)

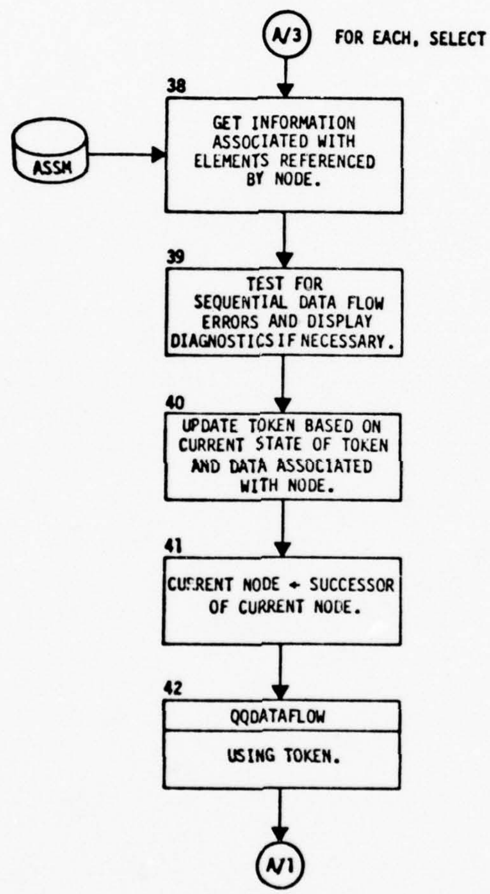



Figure 3-39 Data Flow Analysis (QQDATAFLOW) (Continued)

### 3.4.11 List Permission (QQLPERM)

Description

Given a CONTROL_PERMISSION identifier, this module produces the RSL statements for the identifier and all other CONTROL_PERMISSION and EXTENSION_PERMISSION identifiers in the ASSM.

Input

| | | |
|---|---|---|
| PERMISSION_ID | - | Identifier of CONTROL_PERMISSION to be displayed. |
| ASSM | - | CONTROL_PERMISSIONs and EXTENSION_PERMISSIONs. |

Output

| | | |
|---|---|---|
| RSL TEXT | - | List of permissions. |

Processing

Figure 3-39.1 contains the functional flow diagram for this module.

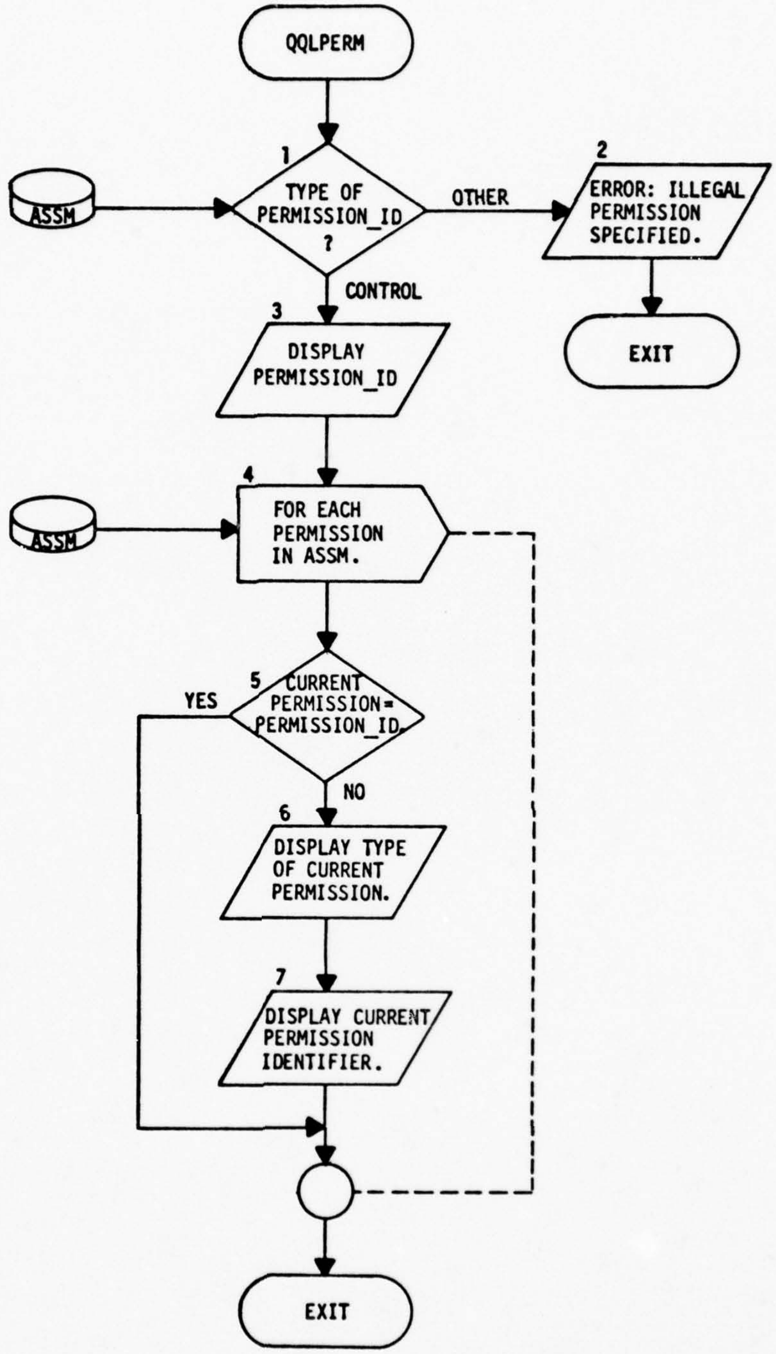



Figure 3-39.1 List Permission (QQLPERM)

### 3.4.12 Plot Structures (QQPLOT)

**Description**

This module generates a CALCOMP plot for each element in the user specified independent set that has a structure.

**Input**

| | | |
|---|---|---|
| INDEPENDENT_SET | - | Collection of elements to be plotted. |
| ASSM | - | Element structures. |
| WIDTH | - | Width of plot. |
| HEIGHT | - | Height of plot. |

**Output**

| | | |
|---|---|---|
| CALCOMP PLOT | - | Plot of each element with a structure. |
| PLOT COUNT | - | Display of number of generated plots. |

**Processing**

Figure 3-39.2 contains the functional flow diagram for this module.

**Procedure References**

The following correlates the functional processing elements in Figure 3-39.2 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [3] | - | XXCNET |

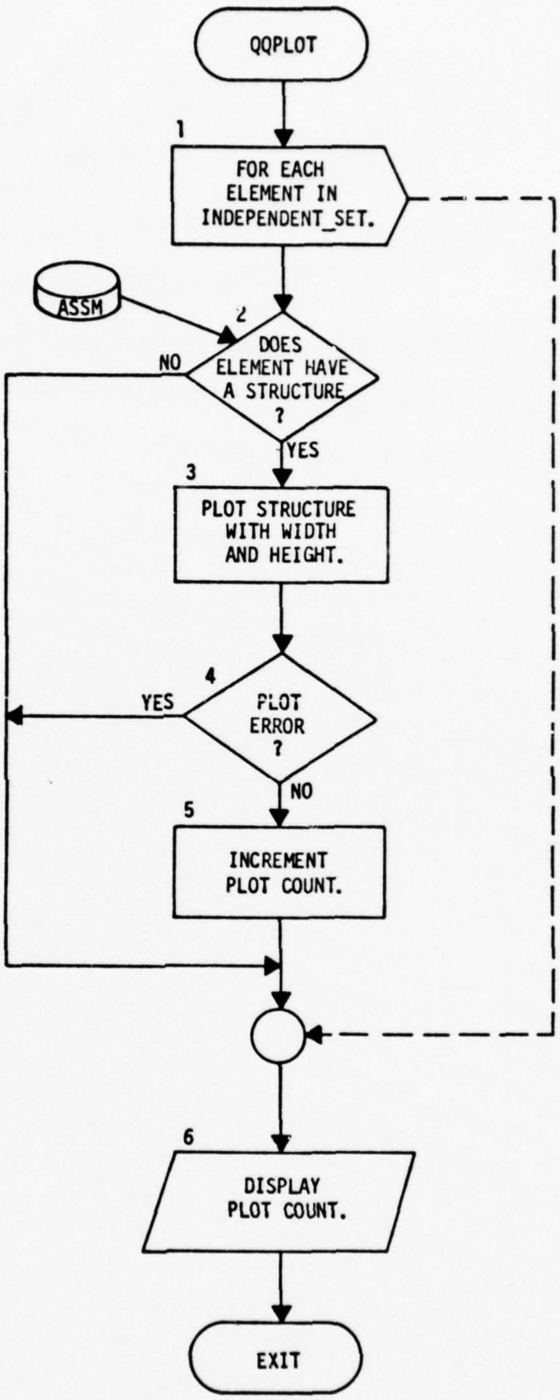



Figure 3-39.2 Plot Structures (QQPLOT)

## 3.5 SIMULATOR GENERATION (SIMGEN)

Description

The Simulator Generation function constructs a Simulator Program which models the software requirements specified in the ASSM. The organization of the Simulator Program source code is shown in Section 3.5.6. The functional components of the Simulator Program and their interrelationships are detailed in Section 4.0. The Simulator Generation function also constructs a Simulator Post Processor Program if an analytic simulator is constructed. The Simulator Post Processor Program is described in Section 5.0.

The Simulator Program components can be separated into three basic groups on the basis of their functions:

a) System Environment and Threat Simulation (SETS)

b) Requirements Modeling

R_NET Model Procedures
SUBNET Model Procedures
ALPHA Model Procedures

c) Simulation Support

Simulator Executive
Simulator Event Management
Simulator Data Management
Simulator Initialization

SETS is the driver for the software requirements models. The production of the source code for the SETS functions is performed independently of the Simulator Generation function and externally to REVS.

The components that perform the requirements modeling functions are termed requirements dependent since these procedures are intended to simulate the behavior of software which meets the requirements specified in the ASSM. The requirements modeling procedures must be created in accordance with the ASSM contents and, consequently, involve a translation from the information content of the ASSM to executable source code.

The simulation support functions are composed of a mix of requirements independent (since they do not depend directly on the contents of the ASSM) and requirements dependent source code. The framework of the Simulator Executive is independent of the ASSM content, while the detailed scheduling and invocation of requirements models is requirements dependent. Likewise, the framework for the Simulator Data Manager and Simulator Initialization remain constant for any ASSM. There is, however, a considerable amount of source code which must be tailored to the ASSM content for each of these functions. The Simulator Event Manager, in contrast, is entirely requirements independent.

The primary functions of Simulator Generation then are to:

- Translate necessary ASSM elements, attributes, and relationships into the requirements modeling procedures and requirements dependent portions of the simulation support functions.
- Consolidate the generated requirements dependent source code with the requirements independent SETS and simulation support components into a compilable source program.
- Compile the simulator source program and linkage edit the object to form a Simulator Program load module.
- Compile the simulator post processor source program and linkage edit the object to form a Simulator Post Processor Program load module.

The Simulator Generation function is performed in five processing phases:

- User Input Control Processing - parses the user input RCL.
- Requirements Analysis - checks for requirements data base consistency and generates linked lists of ASSM elements required in Translation.
- Translation - generates PDL 2 source statements representing the requirements specifications in the ASSM.
- Consolidation - combines the requirements dependent PDL 2 source statements and the requirements independent source modules for input to the PDL 2 compiler.

- Compilation - calls a REVS Executive utility which invokes the PDL 2 compiler and linkage editor to generate load modules for the Simulator Program and Simulator Post Processor Program. In the current implementation, the Executive utility sets a JSL variable which causes the compiler and linkage editor to be executed after completion of REVS execution (see Section 6.3).

Input

ASSM - Simulator Generation accesses elements, attributes, and relationships as well as R_NET and SUBNET structures.

REQUIREMENTS INDEPENDENT SOURCE FILE (RISF) - The RISF contains source code needed to build components of the Simulator Program that are independent of the particular requirements model being generated (See Section 7.2.3).

SETS DEFINITION FILE (SDF) - The SDF contains the SETS source code (See Section 7.2.3).

USER RCL - Simulator Generation control statements. These statements specify what type of simulator is to be generated (BETA or GAMMA) and which R_NETs are to be included in the simulator.

Output

EVENT/ENABLEMENT DEFINITION FILE (EEDF) - The EEDF contains the information necessary to initialize enablement control tables that are used by the Simulator Program to associate procedure enablements with EVENTs, INPUT_INTERFACEs, and OUTPUT_INTERFACEs. The EEDF contains an Event Definition Record for each EVENT, INPUT_INTERFACE, and OUTPUT_INTERFACE referenced by the software requirements model. The EEDF also contains the data and time SIMGEN was executed, the ID given by the user and names of the PERFORMANCE_REQUIREMENTs.

Event Definition Record

Event Number

Event Name

Event Type (Immediate, Delayed, Input Interface, Output Interface)

Delay

Pointer to First Dependent Procedure

Pointer to Last Dependent Procedure

LOAD MODULE FILE (LMF) - Object load module for the Simulator Program.

PERFORMANCE REQUIREMENT SOURCE FILE (PRSF) - A PDL 2 text file containing the source code for the Simulator Post Processor Program.

Local Data

CONSTANT DECLARATION FILE (CDF) - A PDL 2 text file containing source code for the requirements dependent constant declarations to be included in the Simulator Program.

COMPILE FILE (CF) - A PDL 2 text file containing the source code for the entire Simulator Program. The CF is input to the PDL 2 compiler. During SIMGEN execution, the CF is also used as a scratch file.

PROCEDURE DECLARATION SOURCE FILE (PDSF) - A PDL 2 text file containing the source code for requirements dependent procedures.

PROCEDURE SCHEDULER SOURCE FILE (PSSF) - A PDL 2 text file containing procedure enablement source code to be inserted in the Procedure Scheduler procedure of the Simulator Program.

SIMULATOR TRANSLATION LIST (STL) - A multi-level linked list of ASSM pointers that define and describe the ASSM data to be translated by SIMGEN. The STL contains the following sublists:

- ALPHA (ALFALIST)
- ENTITY_CLASS (CLSSLIST)
- EVENT (EVNTLIST)
- FILE (FILELIST)
- INPUT_INTERFACE (INLIST)
- OUTPUT_INTERFACE (OUTLIST)
- R_NET (RNETLIST)

- SUBNET (SNETLIST)
- Simple DATA (SDATLIST)
- VALIDATION_POINT (VALLIST)
- PERFORMANCE_REQUIREMENT (PRLIST)

The structure of STL and its sublists are shown in Figures 3-40 through 3-47.

TYPE DECLARATION SOURCE FILE (TDSF) - A PDL 2 text file containing source code for the requirements dependent type declarations to be included in the Simulator Program.

VARIABLE DECLARATION SOURCE FILE (VDSF) - A PDL 2 text file containing source code for the requirements dependent variable declarations to be included in the Simulator Program.

Processing

Simulator Generation processing is shown in Figure 3-48. Additional commentary for some of the processing steps is given below.

[1] - Necessary control variables are initialized. No values for simulator type (BETA or GAMMA) or the scope of the simulator (which R_NETs are to be included) are assumed.

[2] - The user input RCL is parsed. (See the REVS Users Manual [3] for the definition of the SIMGEN RCL.) Both the simulator type (BETA or GAMMA) and the scope of the simulator (which R_NETs to include or exclude) are required inputs.

[9-10] - The Requirements Analysis and Data Extraction Function (RADX) is called through the REVS Executive interface procedure XXRADX to perform an analysis of the R_NETs to be included in the simulator build. The RADX function also constructs the Simulator Translation List (STL), a linked list of ASSM element pointers, which is used to control the phases of translation.

[11] - The date and time of the SIMGEN execution, along with the ID supplied by the user and the names of PERFORMANCE_REQUIREMENTs included in the simulator build are recorded on the EEDF for later use by SIMDA and by the Simulator Program.

[12-19] - The steps of translating the ASSM content into executable PDL 2 source code are executed. Each of these steps is further described in the following sections.

[20] - To conserve memory space, the space occupied by the Simulator Translation List (STL) is released.

[21-23] - If no fatal errors were encountered in any of the translation steps, the consolidation phase will combine all of the requirements dependent source code built during translation with the requirements independent source code and SETS modules to construct a compile file (CF).

Procedure References

The following list correlates the functional processing steps shown in Figure 3-48 with the REVS procedures in which the processing is performed.

[2] - GGPARSER

[9] - XXRADX

[11] - GGPRIMEDF

[12] - GGTRDATA

[13] - GGTREVNT

[15] - GGTRVP

[16] - GGTRPR

[17] - GGTRVAL

[18] - GGTRALFA

| | | |
|---|---|---|
| [19] | - | GGTRRNET |
| [20] | - | GG8DISPOSE |
| [21] | - | GGCONSOL |
| [22] | - | GGCOMPIL |

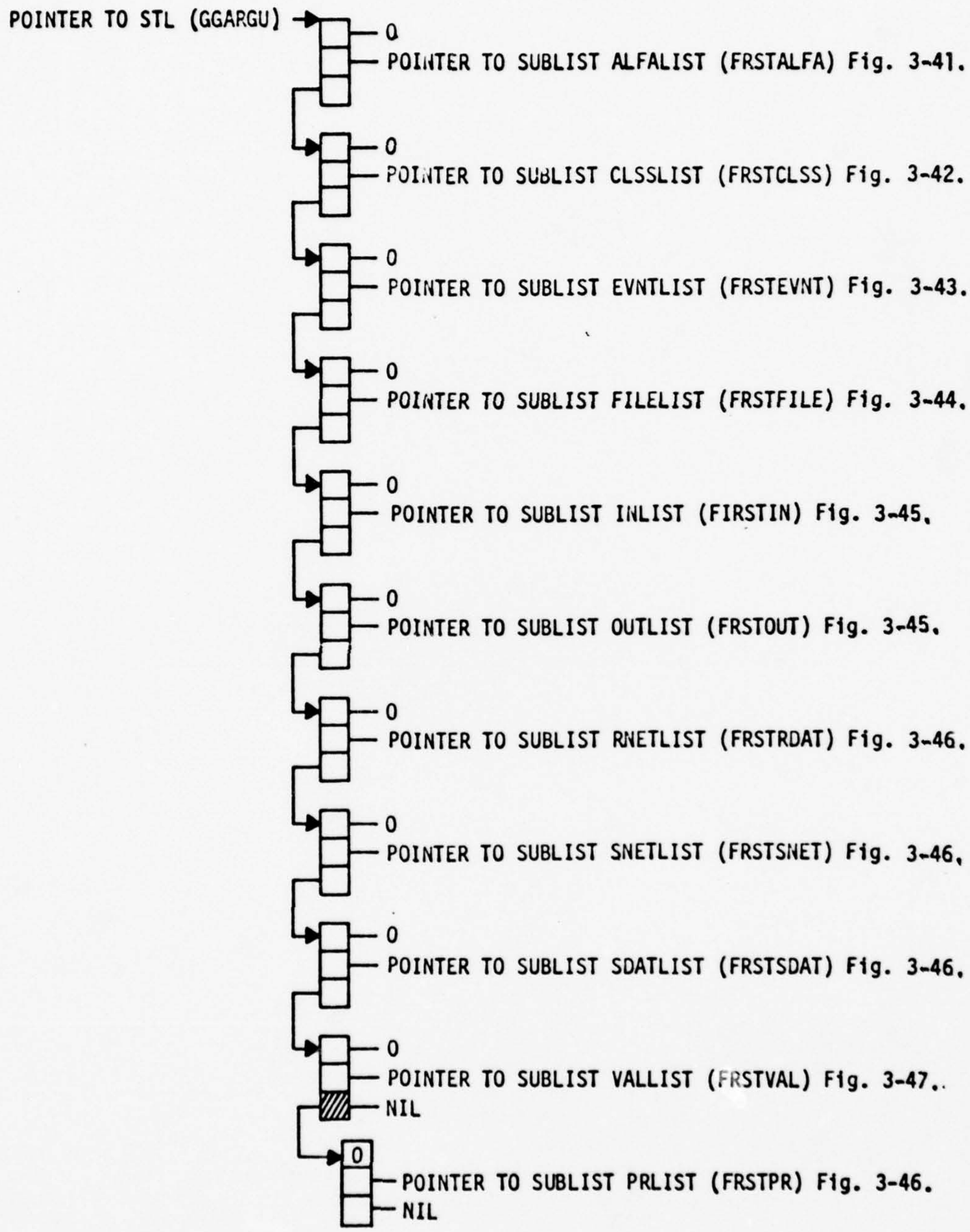



Figure 3-40 Overview of Simulator Translation List (STL)

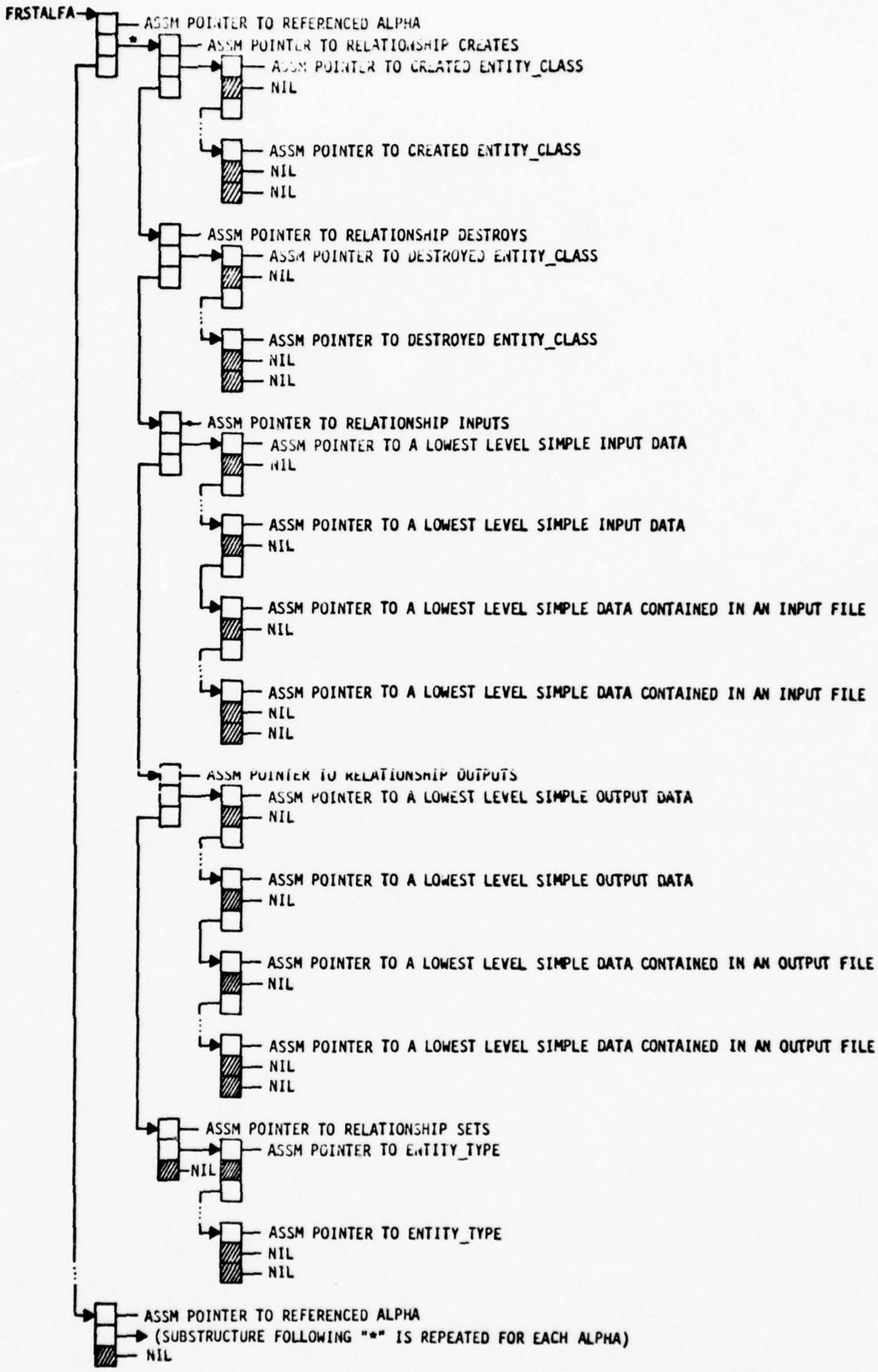



Figure 3-41 ALPHA List (ALFALIST)

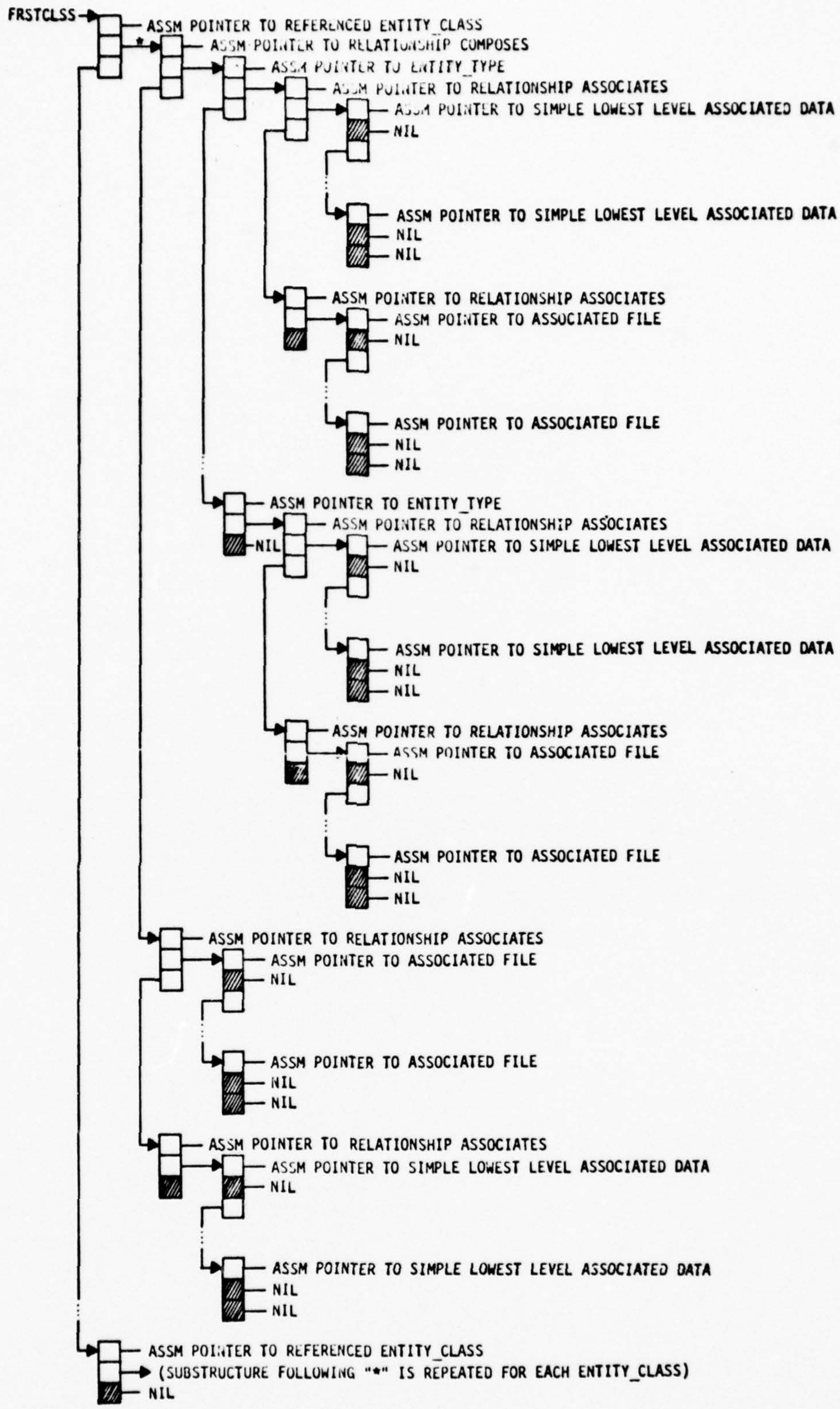



Figure 3-42 ENTITY_CLASS List (CLSSLIST)

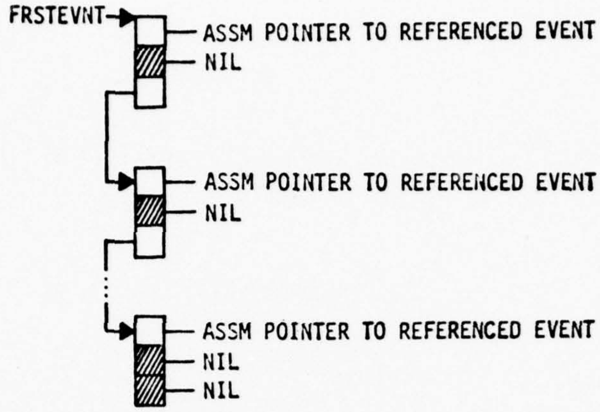



Figure 3-43 STL Sublist (EVNTLIST)

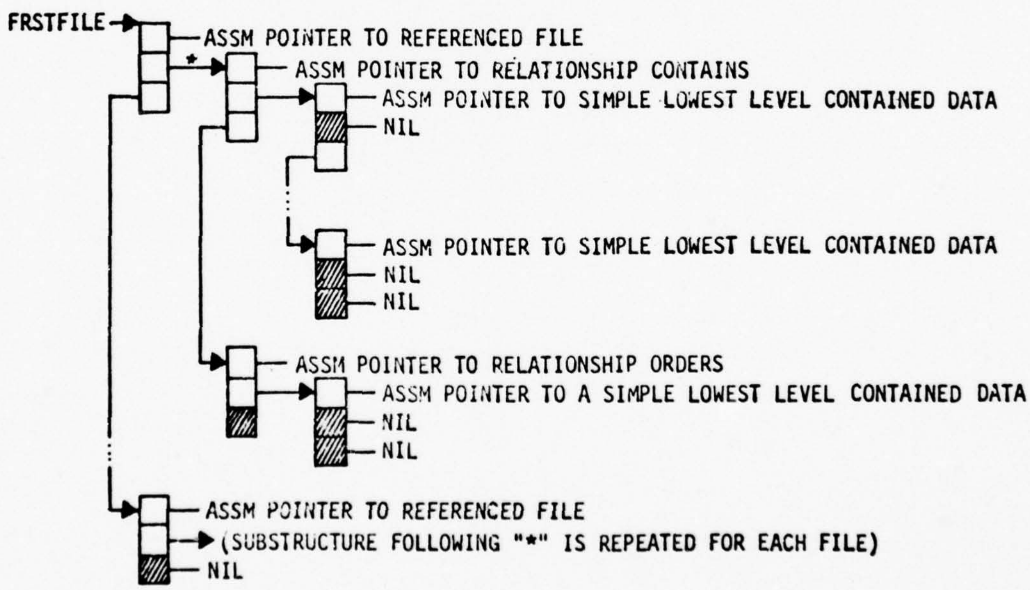



Figure 3-44 FILE List (FILELIST)

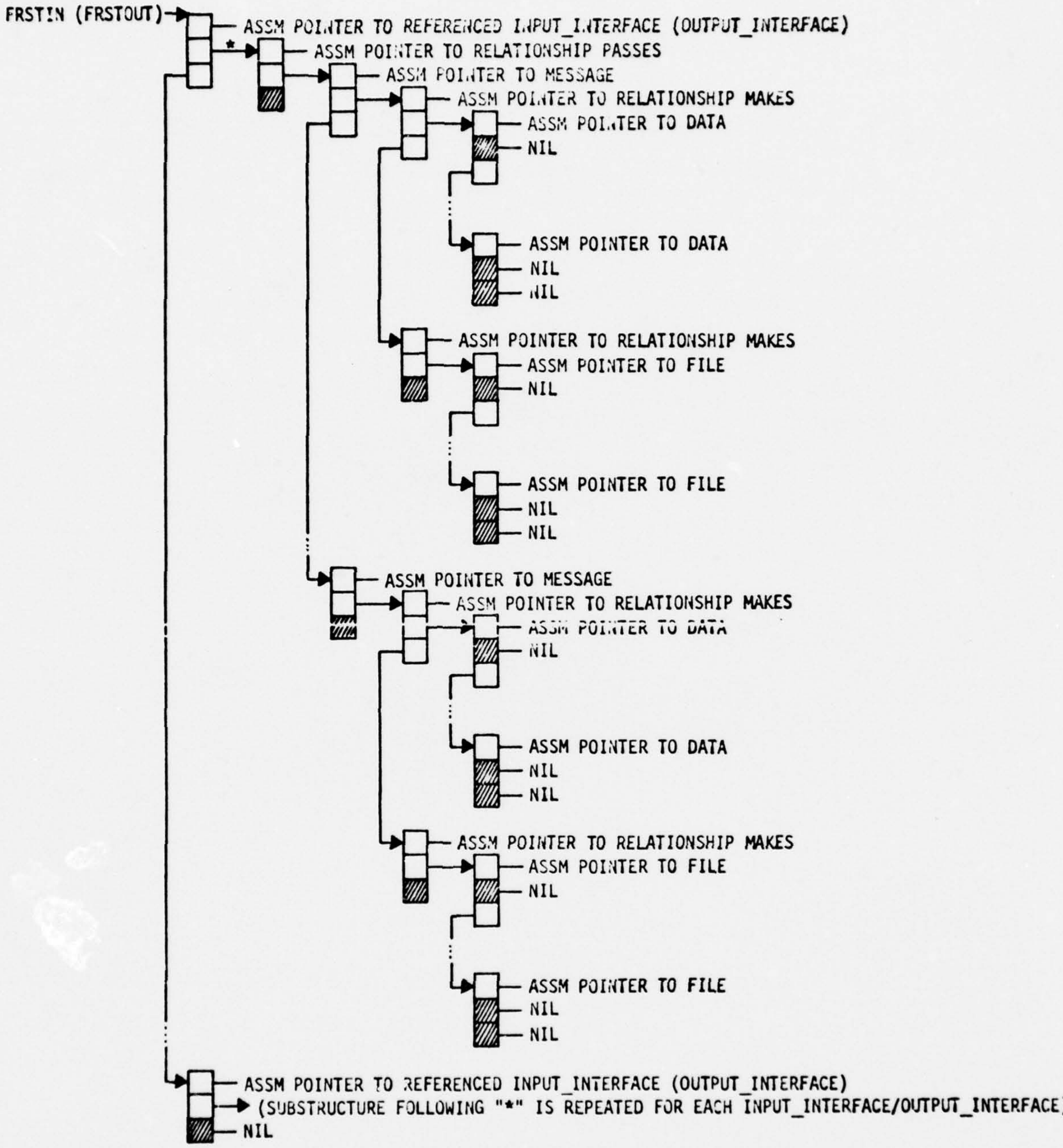



Figure 3-45 INPUT and OUTPUT_INTERFACE List (INLIST, OUTLIST)

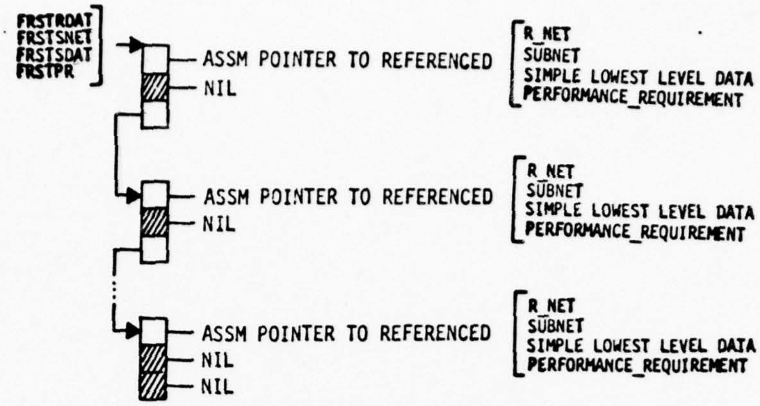



Figure 3-46 R_NET, SUBNET, and Simple DATA List (RNETLIST, SNETLIST and SDATLIST)

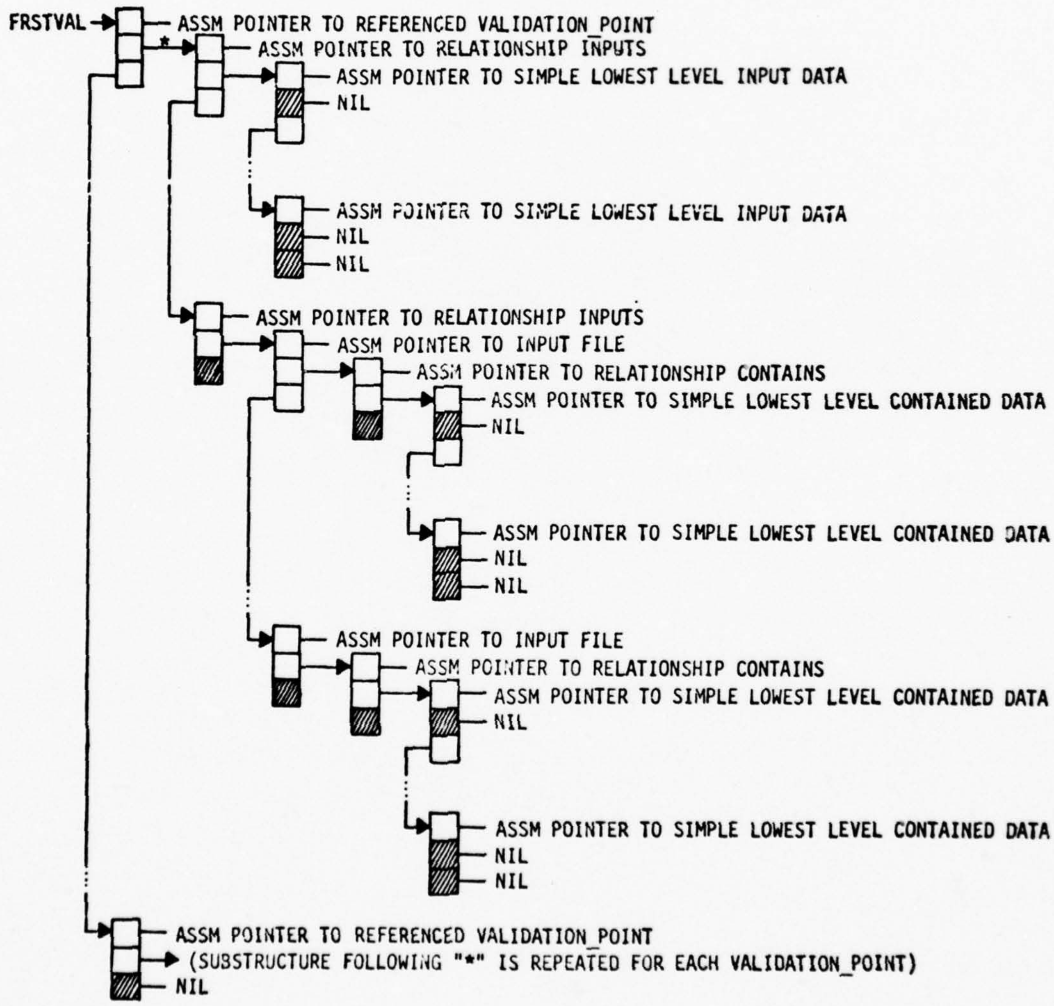



Figure 3-47 VALIDATION_POINT List (VALLIST)

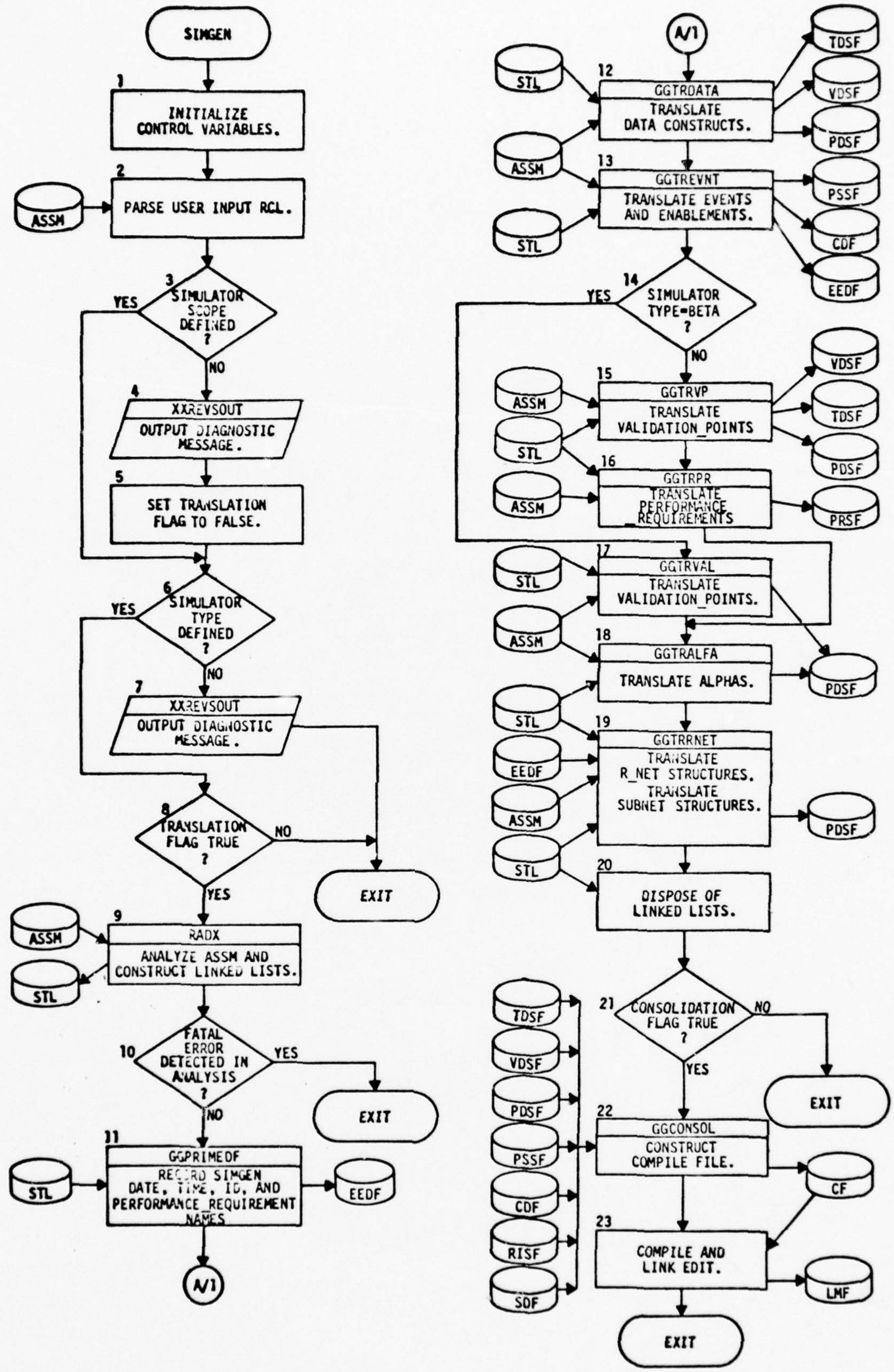



Figure 3-48 Simulator Generation (SIMGEN)

### 3.5.1 Data Translation (GGTRDATA)

Description

The Data Translation module of the Simulator Generation function examines the repetitive and simple data items in its input lists and generates the PDL 2 declaration statements to define Data Management structures of the required nature and PDL 2 procedures to access those Data Management structures. Section 4.4 gives a complete description of the Data Management functions that are provided.

Figure 3-49 illustrates the functional linkages used to model the Repetitive Data Sets (RDS) structure and used to access it from the R_NETs and ALPHAs. RDS's are stratified into four levels of complexity - class, data-set, instance, and data-item. These levels are generalized to cover the many special cases of data objects that REVS supports. A class is a generalization of ENTITY_CLASS and is defined as a collection of data-sets. Every data-set belongs to one and only one class. A data-set is a generalization of FILE and is defined as a collection of instances. A FILE, and INTERFACE, and an ENTITY_TYPE are all data-sets each belonging to a class. Two special pseudo-classes are generated, one containing all FILEs and the other containing all INTERFACEs. An instance is an occurrence of a collection of DATA and/or FILES belonging to a data-set. In the case of FILEs and ENTITY_TYPEs, all instances belonging to a data-set are identical but an INTERFACE may pass different MESSAGEs (i.e., own different kinds of instances). Since the number of instances belonging to a particular data-set is variable, instance storage is dynamically allocated and released and instance access is via doubly linked list structures. The R_NET or ALPHA requests data manager action on instances with a number of different high level statements such as CREATE, DESTROY, and SELECT. Finally, at the lowest level, individual data items are the smallest quantum of information and may be 'simple' (unattached to a Repetitive Data Set) or may be a member of an instance. If DATA is associated with an RDS, the R_NET or ALPHA model gains access to it via the SELECT or FOR EACH high level commands if an instance is located. If two instances cannot be selected at once, DATA may belong to both -- as in the case of instances of two ENTITY_TYPEs owned by the same ENTITY_CLASS.

These generalized data elements are translated into three different PDL 2 constructions -- type declarations, variable declarations, and executable procedures. These constitute the three outputs of the Data Translation module.

Each data-set and class is declared in a PDL 2 type declaration as one element of an enumerated type, EE7DSLST. This enumerated type is declared as the index into the Data-Set/Class Description Array (EE9DS) which, at simulator execution time, describes all data-sets and classes. The elements of the Data-Set/Class Description Array are two variants of one type, EE7DSTYP. One variant is generically named a Data-Set Description Block because such an entry contains an execution time description of a data-set and points to the linked list of dynamically allocated instances which the data-set owns. The other variant is called a Class Description Block because it contains, at simulator execution time, a description of a class and indicates which data-sets are owned by the Class. Section 4.4 contains a detailed breakdown of all data fields in these two entry types.

Each instance is declared in a PDL 2 type declaration as one element of an enumerated type, EE7INLST and as a variant of a 'super' record description, EE7INTYP. Each data item of an instance is declared as a field within the instance variant of EE7INTYP. Thus all dynamically allocated instance records are of the same PDL 2 type with variants depending on the instance type. A static variable of type specified in the ASSM is declared for every data item (simple or repetitive). The user accesses a data item by operating on the corresponding static variable using PDL 2 statements. For a repetitive data item, an instance must be selected (data values transferred from the dynamic instance record to the static variables) before the static variables contain valid information.

Most of the PDL 2 procedures which are created during Data Translation manipulate the dynamically allocated instance records and/or static variables in accordance with higher level Data Management requests. These procedures create a dynamic instance record of a requested variant type (EE8NEWI), destroy a dynamic instance record of a requested variant type (EE8DISI), transfer data items to and from a dynamic instance record (EE8XIO and EE8XII),

order dynamic instance records in a data-set (EE8KYCMP), and transfer data items between dynamic instance records (EE8CPYIN). The remaining generated PDL 2 procedures initialize the static instance variables (EE8ININ, EE8INCL, EE8IIFAC, and EE8LOCAL) or initialize the Data-Set Description Blocks (EE8SETUP and EE8FORM).

The Data Translation module, in the course of performing its functions as described above, generates a series of linked lists which are used by other modules in the SIMGEN function. The lists which are generated are as follows: A list of all files used in the simulation (headed by GG9LFILST), a list of all data items used in the simulator (headed by GG9DECLST), a list of all types used in the simulator (headed by GG9TYPLST), a list of all initial values used in the simulator (headed by GG9IVLST), a list of all enumerated types with sublists of enumerated values used in the simulator (headed by GG9ENMLST), and a list of all owners of data or files in the simulator (headed by GG9ØWNRL).

Input

| | | |
|---|---|---|
| ASSM | - | All data element names. |
| SIMPLE DATA LIST (SDATLIST) | - | A simple linked list of all simple (non-repetitive) DATA items to be included in the Simulator Program (See Figure 3-46). |
| FILE LIST (FILELIST) | - | A complex (multi-level) linked list of all FILEs to be included in the Simulator Program with sublists of DATA items CONTAINED IN each FILE (See Figure 3-44). |
| INPUT_INTERFACE LIST (INLIST) | - | A complex (multi-level) linked list of all INPUT_INTERFACEs to be included with sublists of MESSAGEs which are PASSED THROUGH each INTERFACE (See Figure 3-45). |
| OUTPUT_INTERFACE LIST (OUTLIST) | - | A complex (multi-level) linked list of all OUTPUT_INTERFACEs to be included with sublists of MESSAGEs which are PASSED THROUGH each INTERFACE (See Figure 3-45). |
| ENTITY_CLASS LIST (CLSSLIST) | - | A complex (multi-level) linked list of all ENTITY_CLASSes to be included in the Simulator Program (See Figure 3-42). |

Output

| | | |
|---|---|---|
| PROCEDURE DECLARATION SOURCE FILE (PDSF) | - | A text file containing the PDL 2 source statements for the procedures generated by Data Translation. The procedures appear on the file in the following order: EE8XIO, EE8XII, EE8ININ, EE8SETUP, EE8NEWI, EE8DISI, EE8INCL, EE8KYCMP, EE8FORM, EE8CPYIN, EE8IIFAC, and EE8LOCAL. |
| TYPE DECLARATION SOURCE FILE (TDSF) | - | A text file containing the PDL 2 source statements for declaring the type of structures used to model the RDS. The type declarations appear in the following order: EE7DSLST, EE7INLST, user enumerated type declarations (if any), EE7DSTYP, and EE7INTYP. |
| VARIABLE DECLARATION SOURCE | - | A text file containing the PDL 2 source statements for declaring the array and variables used to model the RDS. The variable declarations appear in the following order: EE9DS (Data-Set/Class Description Array), and DATA item variable declarations. |
| FILE LIST (headed by GG9LFILST) | - | A linked list of all FILES processed by the data translation module. Each entry of the list has a sub-list of all the FILE's owners (ENTITY_CLASS or ENTITY_TYPES), the FILE's name (an AASTRING), the FILE's ASSM address, a LOCALITY flag (BOOLEAN) indicating whether the FILE is LOCAL (FALSE) or GLOBAL (TRUE), and an owner-check id (INTEGER) identifying the FILE's owner-check procedure. |
| DATA LIST (headed by GG9DECLST) | - | A linked list of all DATA processed by the Data Translation module. Each entry of the list has a sub-list of all the DATA's owners (FILEs, ENTITY_CLASS, or ENTITY_TYPES), the DATA's name (an AASTRING), the DATA's ASSM address, a pointer into the DATA-TYPE LIST indicating the DATA's PDL2 TYPE, a pointer into the DATA-INITIAL-VALUE LIST indicating the DATA's INITIAL-VALUE, an INITIAL_VALUE status flag (INTEGER) indicating that the DATA's |

INITIAL_VALUE has been looked for in the ASSM (1) or not (0), and an owner-check id (INTEGER) identifying the DATA's owner-check procedure.

| | | |
|---|---|---|
| DATA-TYPE LIST (headed by GG9TYPLST) | - | A linked list of all the DATA-TYPES (PDL 2) processed by the Data Translation module. Each entry of the list has the DATA-TYPE's name (an AASTRING), the DATA-TYPE's ordinal (INTEGER), and a scratch flag (BOOLEAN) used in other modules to indicate whether the DATA-TYPE is used or not. |
| DATA-INITIAL-VALUE LIST (headed by GG9IVLST) | - | A linked list of all the INITIAL_VALUES of DATA processed by the Data Translation module. Each entry of the list has value of the INITIAL_VALUE as an AASTRING. |
| DATA-ENUMERATED-TYPE LIST (headed by GG9ENMLST) | - | A linked list defining all the ENUMERATED-TYPES of DATA processed by the Data Translation module. Each entry of the list has a pointer to the ENUMERATED-TYPE's entry in the DATA-TYPE LIST, a sublist of values composing the ENUMERATED-TYPE's RANGE, and a countered giving the number of values composing the ENUMERATED-TYPE's RANGE. |
| ØWNERS LIST (headed by GG9OWNRL) | - | A linked list of all the DATA and FILE owners processed by the Data Translation module. Each entry of the list has the owner's name (an AASTRING), the owner's ASSM address, and the owner's type (enumerated type designating no type, MESSAGE, FILE, ENTITY_TYPE, and ENTITY_CLASS). |

Processing

Processing performed in the Data Translation module is shown in the flow diagram of Figure 3-50. The following comments refer to processing steps in the flow diagram.

| | | |
|---|---|---|
| [1] | - | Includes such operations as assigning values to variables and arrays for use throughout the Data Translation module, obtaining ASSM addresses for attributes, and generating PDL 2 procedure headers. |

[8] - Implies that the DATA item will be initialized only once at the beginning of simulator execution.

[9] - Implies that the DATA item will be initialized at the beginning of each R_NET executive.

[10] - Includes such operations as initializing first data-set of class name and last data-set of class name.

Procedure References

The following correlates the functional processing elements shown in Figure 3-49 with the REVS procedures which perform the indicated processing.

[1] - GG8DXINI

[2-9] - GG8SIMPLE

[10-35] - GG8FILES

[36] - GG8ICLASS

[37-63, 64-65] - GG8INTRFACES

[66,67] - GG8ECLASS

[68-109] - GG8CLASSES

[110] - GG8DXEND

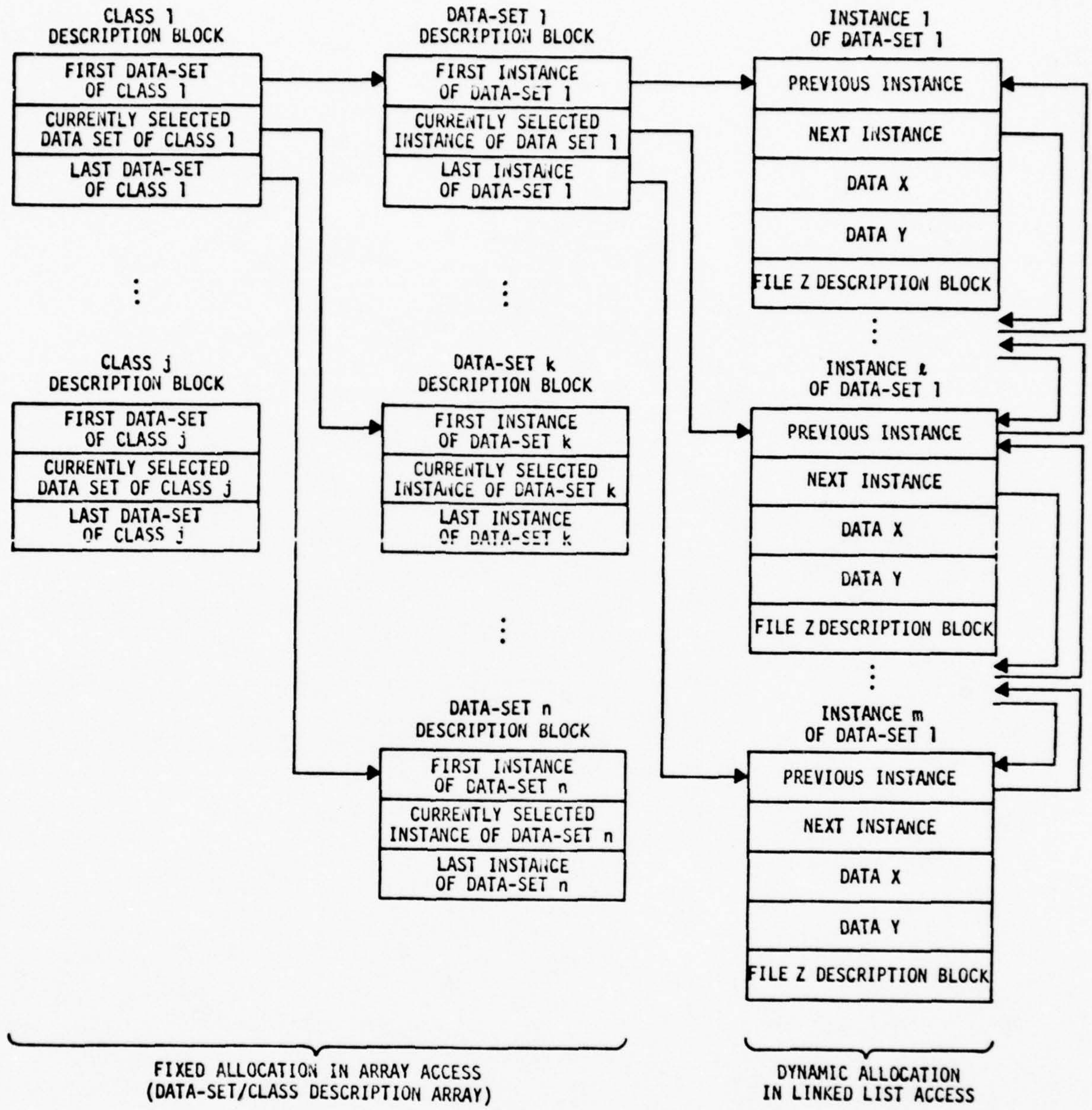



Figure 3-49 RDS Allocation and Access

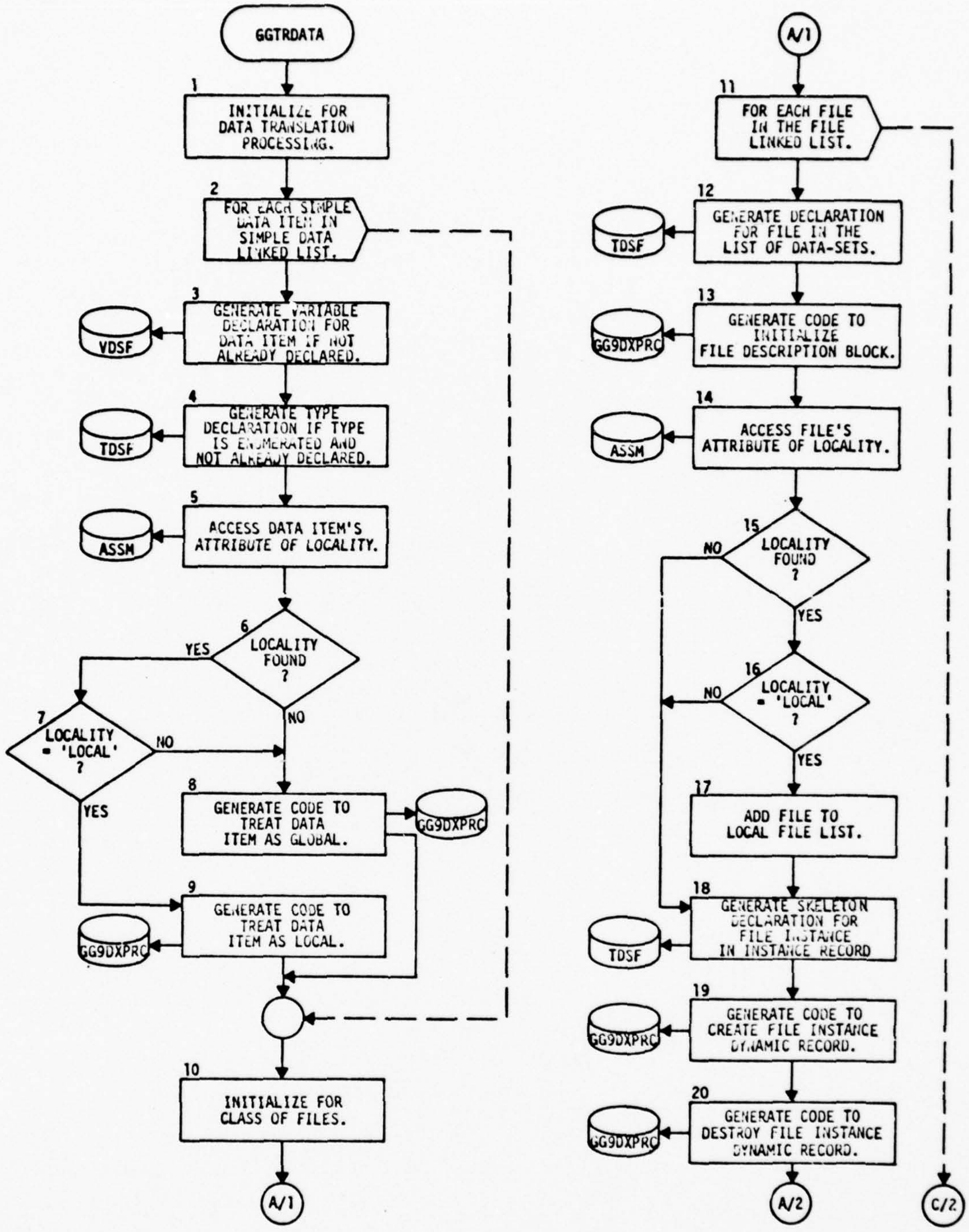



Figure 3-50 Data Translation (GGTRDATA)

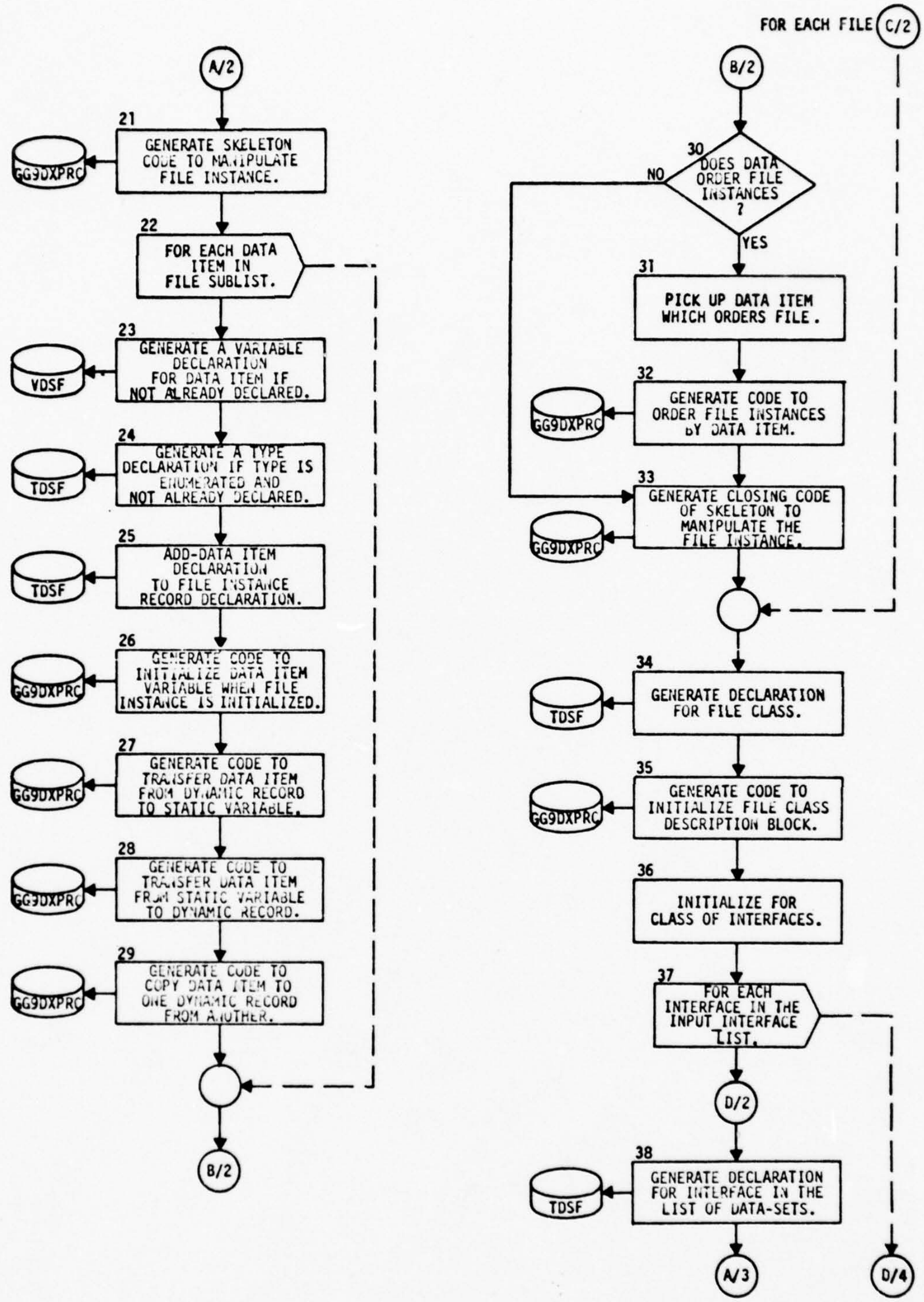



Figure 3-50 Data Translation (GGTRDATA) (Continued)

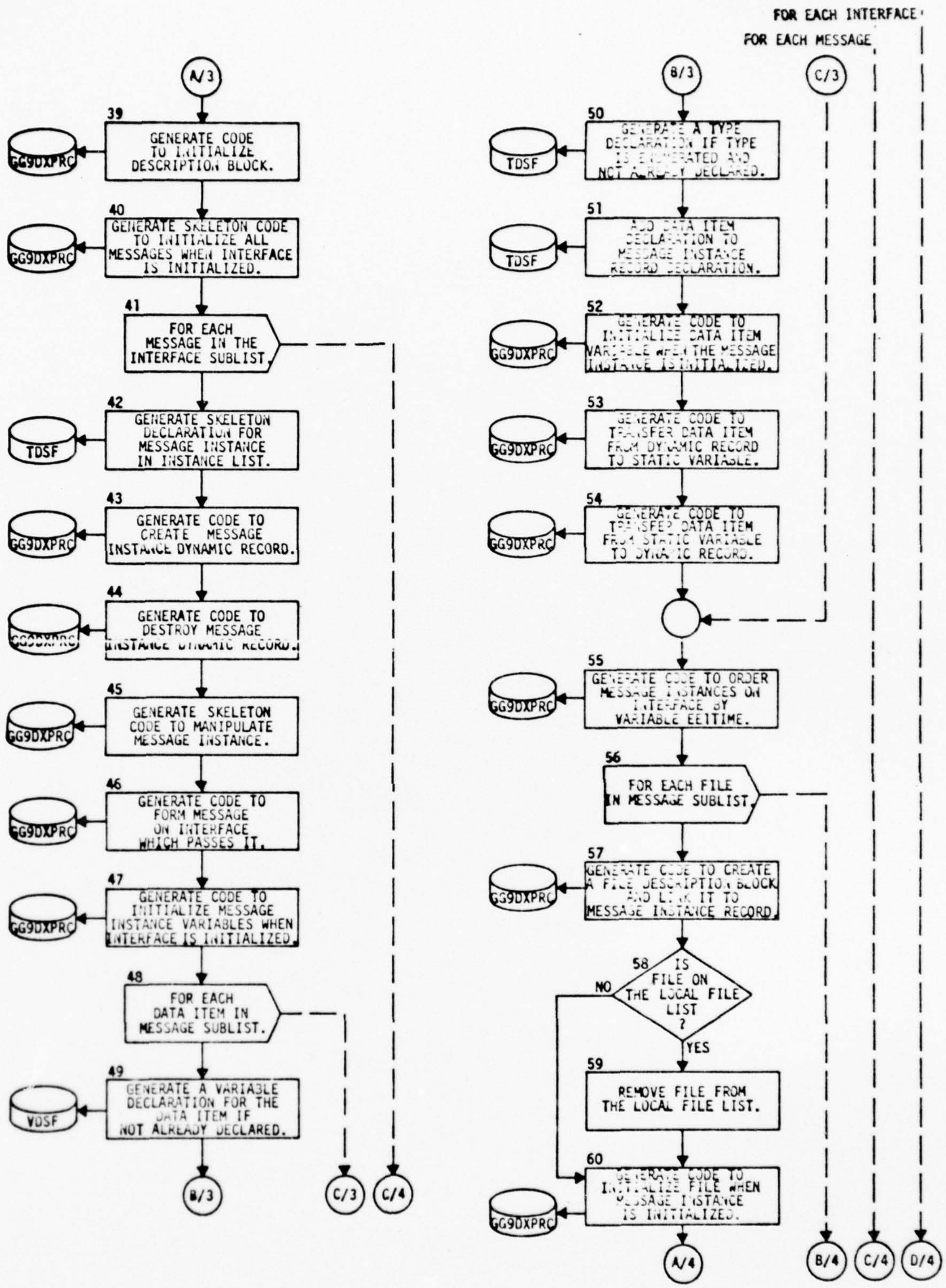



Figure 3-50 Data Translation (GGTRDATA) (Continued)

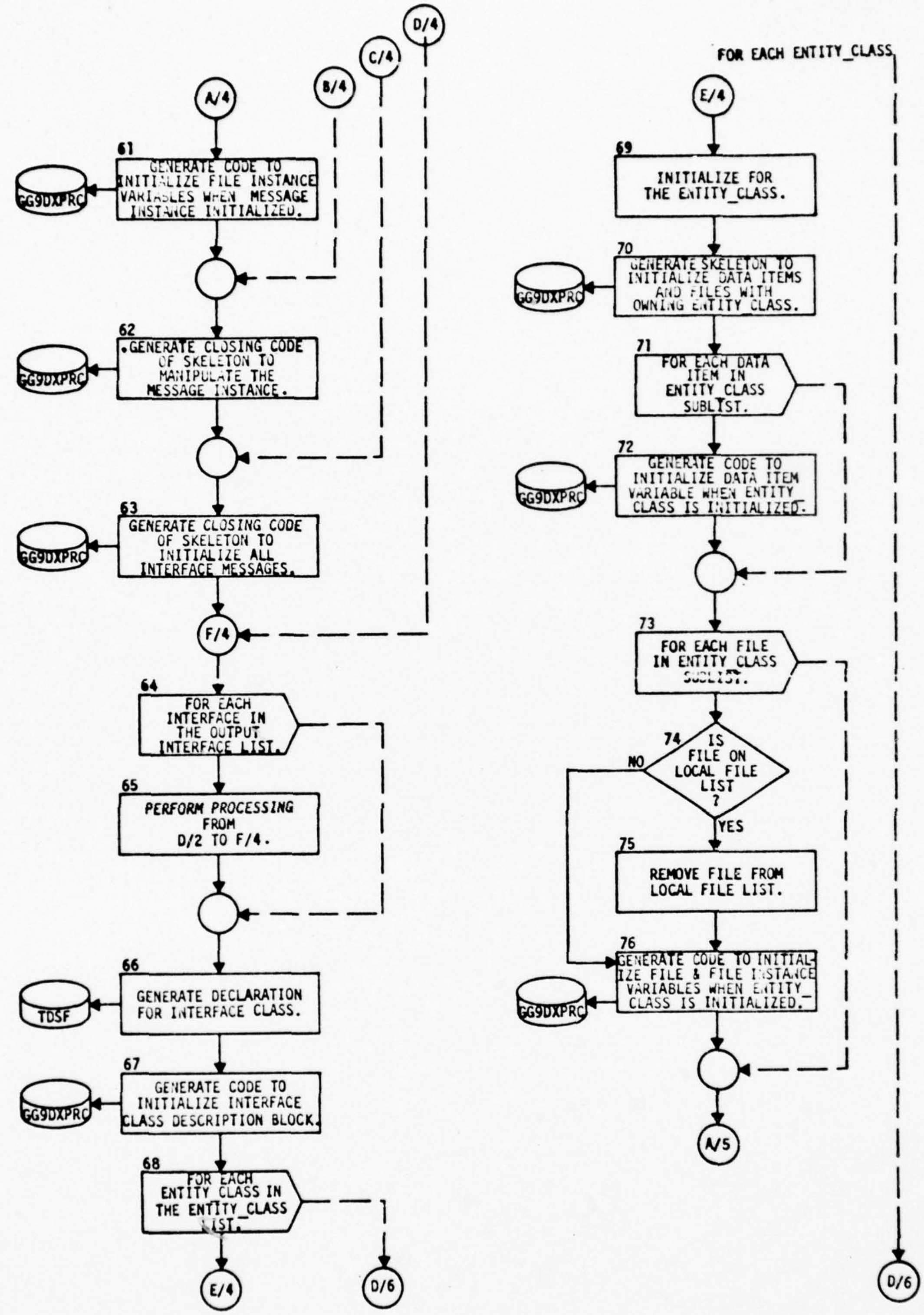



Figure 3-50 Data Translation (GGTRDATA) (Continued)

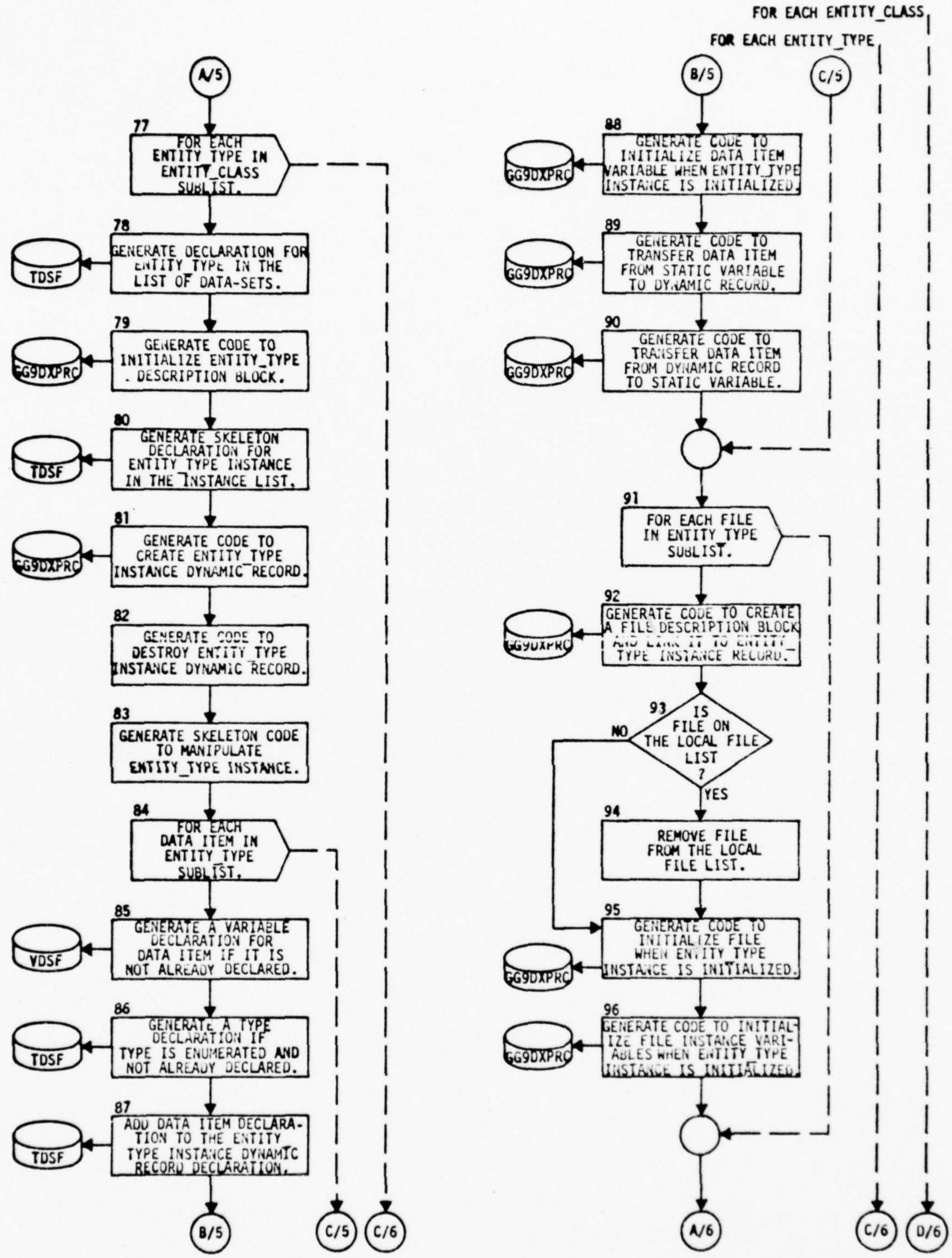



Figure 3-50 Data Translation (GGTRDATA) (Continued)

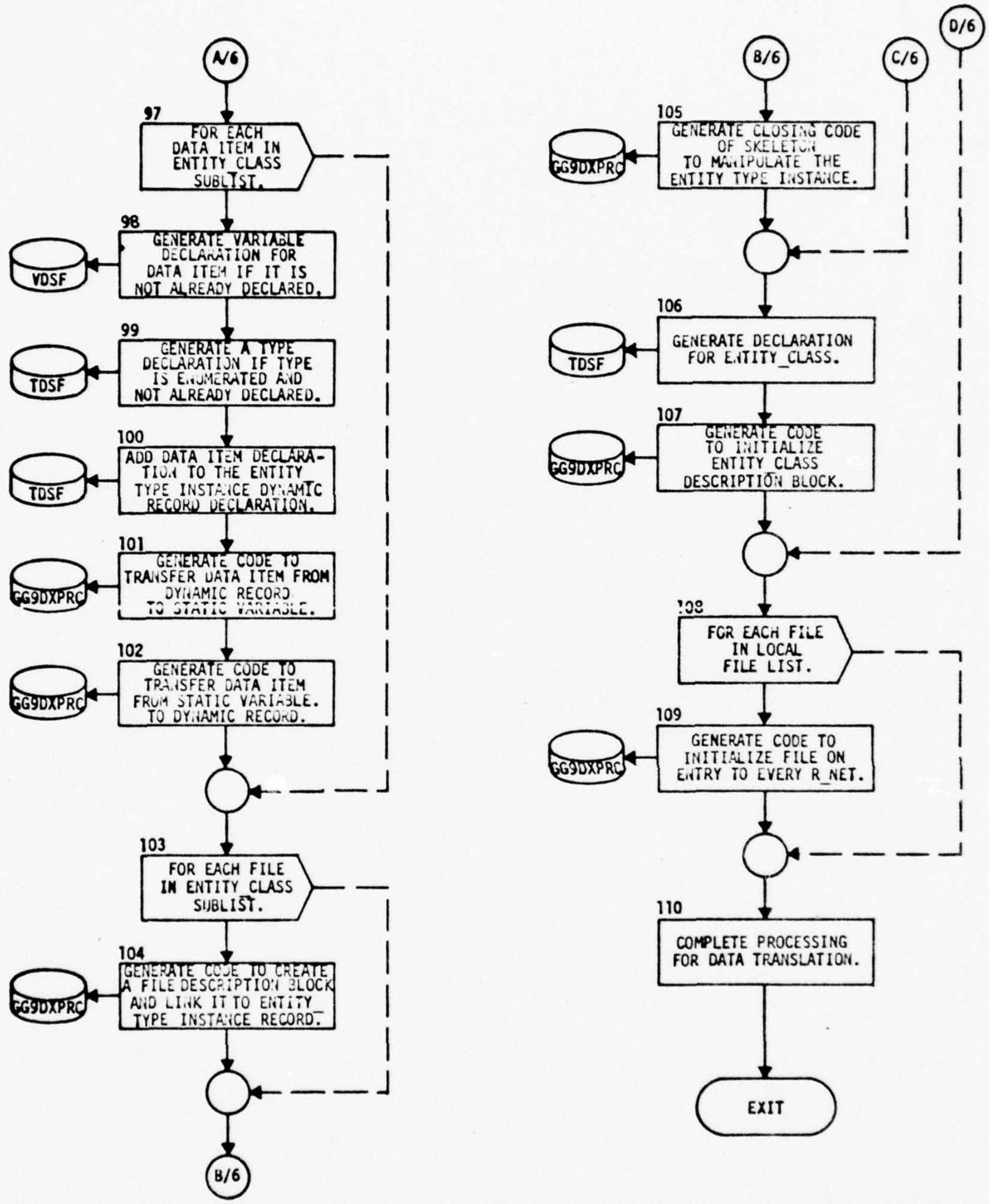



Figure 3-50 Data Translation (GGTRDATA) (Continued)

### 3.5.2 Event/Enablement Translation (GGTREVNT)

Event/Enablement Translation retrieves ASSM information related to EVENTs and INTERFACEs and uses it

- to define the input values and length of procedure enablement control tables used by the Simulator Program, and
- to generate source code for the Simulator Program's Procedure Scheduler (EESCHED).

The control table inputs are stored in the Event/Enablement Definition File (EEDF) which is used by EEINITIAL, the Simulator Initialization procedure, to load the EEVLIST table described below. The control table lengths are output on the Constant Declaration File (CDF) and the Procedure Scheduler source code is output on the Procedure Scheduler Source File (PSSF). The contents of both of these files are copied directly to the Compile File during the consolidation phase of Simulator Generation.

The procedure enablement control tables consist of the Event List, EEVLIST, and the Event Dependency List, EEDEPLST. EEVLIST is an array of records each of which describes an EVENT or INTERFACE to be included in the Simulator. Each record also includes EEDEPLST indices for the first element and the last element in the group of EEDEPLST elements enabled by this EVENT (or INTERFACE). The EEVLIST structure is illustrated in the example below.

EEDEPLST is an array of logical enablement variables each of which is associated with an R_NET, special procedure or SETS module. As illustrated in the example below, the elements of EEDEPLST are grouped according to EVENT (INTERFACE), with each element in the group corresponding to an R_NET or special procedure (SETS module) dependent on that EVENT (INTERFACE) for enablement. Note that more than one EEDEPLST element may be associated with a given R_NET if the R_NET is enabled by one of several EVENTs. The logical variables in EEDEPLST are set TRUE by the Simulator Executive to effect enablement of the corresponding R_NET, special procedure or SETS module.

The generated Procedure Scheduler source code calls enabled procedures and resets to FALSE the enablement variables in EEDEPLST.

Input

ASSM

| | | |
|---|---|---|
| EVENT LIST (EVNTLIST) | - | A multi-level linked list of EVENTs referenced by the Simulator Program (See Figure 3-43). EVNTLIST is a sublist of the Simulator Translation List. |
| INPUT_INTERFACE LIST (INLIST) | - | A multi-level linked list of INPUT_INTERFACEs referenced by the Simulator Program (See Figure 3-45). INLIST is a sublist of the STL. |
| OUTPUT_INTERFACE LIST (OUTLIST) | - | A multi-level linked list of OUTPUT_INTERFACEs referenced by the Simulator Program (See Figure 3-45). OUTLIST is a sublist of the STL. |
| R_NET LIST (RNETLIST) | - | A simple linked list of R_NETs referenced by the Simulator Program (See Figure 3-46). RNETLIST is a sublist of the STL. |

Output

| | | |
|---|---|---|
| EVENT/ENABLEMENT DEFINITION FILE (EEDF) | - | The EEDF contains the information necessary to initialize enablement control tables which associate EVENTs and procedure enablements. It contains an Event Definition Record for every EVENT or INTERFACE referenced by the requirements model. The content of these records is defined in Section 3.5. |
| PROCEDURE SCHEDULER SOURCE FILE (PSSF) | - | A PDL 2 text file containing procedure enablement source code, defined below, that is inserted in the Procedure Scheduler procedure EESCHED. |
| CONSTANT DECLARATION FILE (CDF) | - | A PDL 2 text file containing constant declarations for EEVMAX (length of EEVLIST) and for EEMAXDEP (length of EEDEPLST). |

Processing

Event/Enablement Translation processing is illustrated by the following example. Assume RSL relationships as follows:

```
INPUTFACE ENABLES R_NET1.
EVENT1 ENABLES R_NET2.
EVENT1 ENABLES R_NET3.
DATA1 DELAYS EVENT1.
EVENT2 ENABLES R_NET2.
EVENT2 ENABLES R_NET4.
OUTPUTFACE CONNECTS RADAR.
```

These relationships would result in the EEVLIST and EEDEPLST structures shown in Figure 3-51.

Processing is shown in the flow diagram of Figure 3-52. The following comments refer to processing steps in the flow diagram.

[14, 22, 26] - For R_NET1, for example, the generated code would be:

```
IF EEDEPLST (5) THEN BEGIN
  EEDEPLST (5):= FALSE; RNET1
END;
```

The dispatch code for SETS modules and special procedures has the same form.

[16, 21, 25] - The Event Definition Record for EVENT1, for example, is contained in line b of the EEVLIST illustration in Figure 3-50. Event Definition Records for special events and interfaces are shown in lines a and c, respectively.

[27] - For this example, the declarations would be

```
EEVMAX = 7;
EEMAXDEP = 10;
```

**Procedure References**

In the flow diagram of Figure 3-52, procedures are called to perform the indicated processing steps as follows:

| | | |
|---|---|---|
| [14] | - | GGPSSF |
| [16] | - | GGEDFVAR |
| [21] | - | GGEDFVAR |
| [22] | - | GGPSSF |
| [25] | - | GGEDFVAR |
| [26] | - | GGPSSF |

EXAMPLE FORMAT FOR EEVLIST

| | EVENT NO. | EVENT NAME | EVENT TYPE | DELAY | EEDEPLST INDEX TO FIRST DEPENDENT PROCEDURE | EEDEPLST INDEX TO LAST DEPENDENT PROCEDURE |
|---|---|---|---|---|---|---|
| a | 0 | EENGAGE | EEIMMED | 0.0 | 7 | 7 |
| | 1 | EESTOP | EEIMMED | 0.0 | 8 | 8 |
| | 2 | EEDEBUG | EEIMMED | 0.0 | 9 | 9 |
| | 3 | EEVALID | EEIMMED | 0.0 | 10 | 10 |
| b | 4 | EVENT1 | EEDELAY | DATA1 | 1 | 2 |
| | 5 | EVENT2 | EEIMMED | 0.0 | 3 | 4 |
| c | 6 | INPUTFACE | EEINFACE | 0.0 | 5 | 5 |
| | 7 | OUTPUTFACE | EEOUTFACE | 0.0 | 6 | 6 |

EXAMPLE FORMAT FOR EEDEPLST

| ENABLING EVENT/INTERFACE | PROCEDURE(S) ENABLED | EEDEPLST |
|---|---|---|
| EVENT1 | R_NET2 → | FALSE |
| | R_NET3 → | FALSE |
| EVENT2 | R_NET2 → | FALSE |
| | R_NET4 → | FALSE |
| INPUTFACE | R_NET1 → | FALSE |
| OUTPUTFACE | RADAR → | FALSE |
| EENGAGE | EENGAGE → | FALSE |
| EESTOP | EESTOP → | FALSE |
| EEDEBUG | EEDEBUG → | FALSE |
| EEVALID | EEVALID → | FALSE |

Figure 3-51 Example Formats for EEVLIST and EEDEPLST

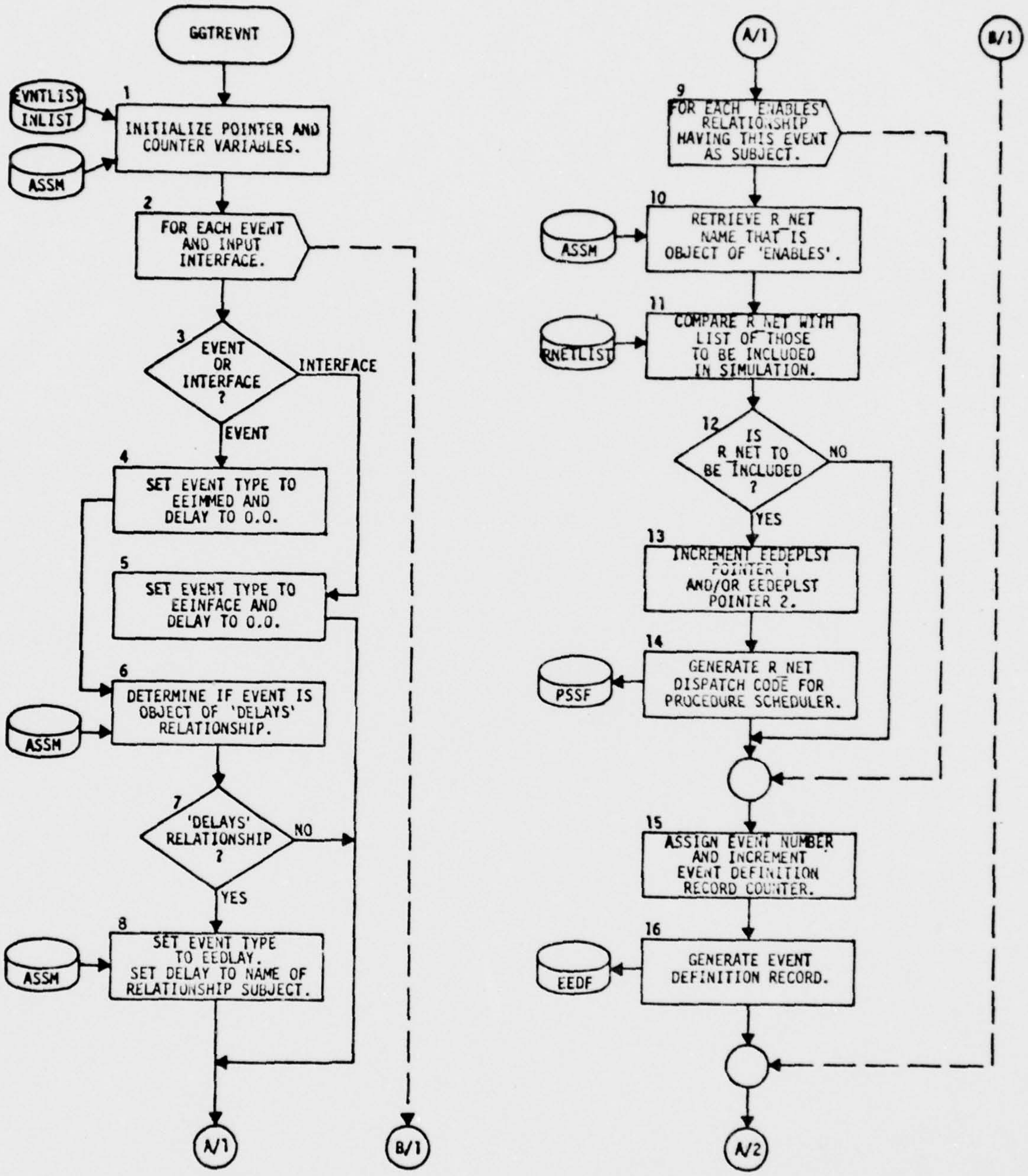



Figure 3-52 Event/Enablement Translation (GGTREVNT)

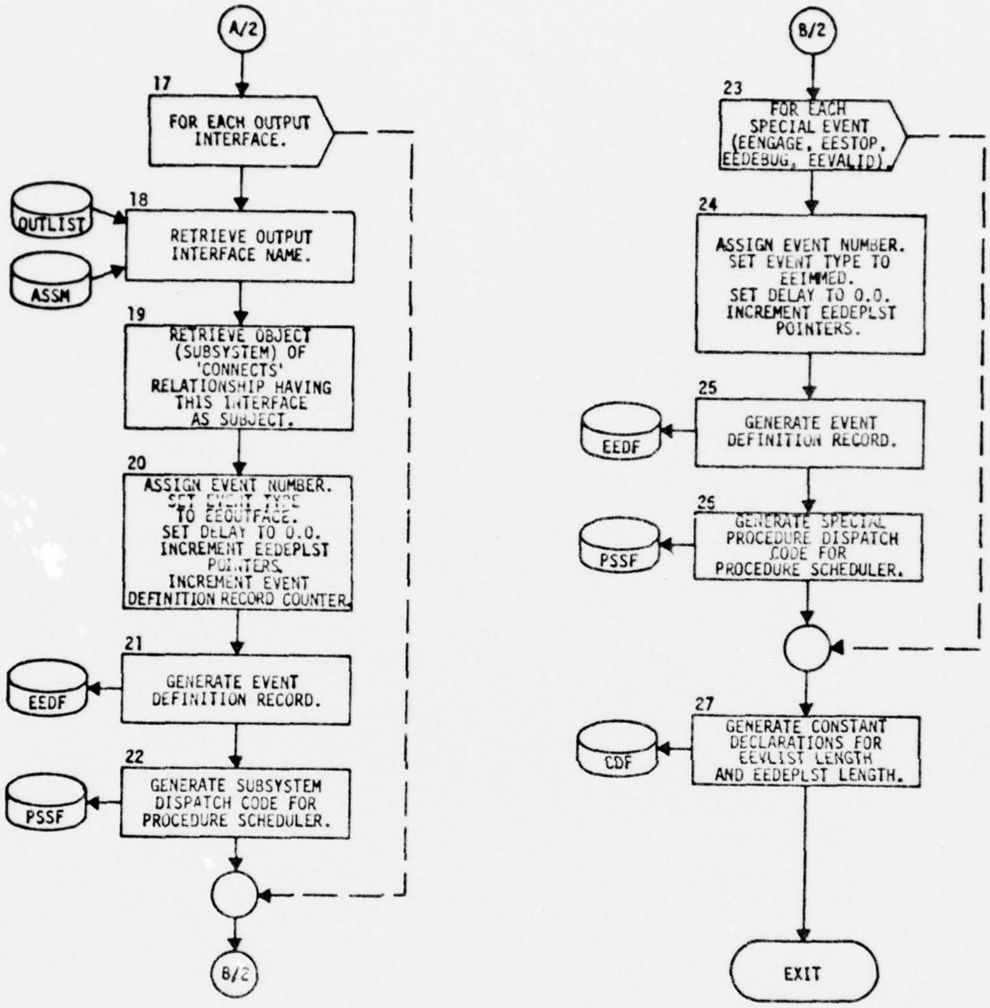



Figure 3-52 Event/Enablement Translation (GGTREVNT) (Continued)

### 3.5.3 Functional Simulator Validation Translation (GGTRVAL)

Description

The Validation Translation module analyzes all VALIDATION_POINTs to be included in the simulator build. It determines the data to be collected at run time and generates the source code to implement recording of the data.

The source code generated for a VALIDATION_POINT consists of calls to Simulator Data Management routines to access instances of repetitive data sets (FILEs), and of PDL 2 output procedure calls to write out the contents of both the FILEs and simple DATA which are RECORDED BY the VALIDATION_POINT.

The generated recording source code for each VALIDATION_POINT is written as a procedure on the PDSF file. During R_NET and SUBNET translation, a call to the appropriate procedure is then generated whenever a VALIDATION_POINT is referenced.

Input

| | | |
|---|---|---|
| VALIDATION_POINT LIST (VALLIST) | - | Linked list of VALIDATION_POINTs with sublists of INPUT simple DATA and FILEs. The FILE lists also have sublists of the DATA CONTAINED in the FILEs (See Figure 3-47). |

Output

| | | |
|---|---|---|
| PROCEDURE DECLARATION SOURCE FILE (PDSF) | - | Recording procedure source code for each VALIDATION_POINT. |

Processing

Processing for the Validation Translation module is shown in Figure 3-53. Additional comments which reference the processing block numbers in that flow diagram are:

| | | |
|---|---|---|
| [2] | - | The header written consists of the PDL 2 procedure and local variable declarations. |
| [4] | - | The source code is a series of PDL 2 WRITELN calls to output data names and values to the file EEVALDAT. |

[6] - The source code written consists of a call to a Data Management routine which saves the current file environment.

[7-9] - The source code uses Data Management routines necessary to sequentially access the instances of the FILE. For each instance, code is generated to write the data names and values to the file EEVALDAT.

[10] - The source code written consists of a call to a Data Management routine which restores the previously saved file environment.

<u>Procedure References</u>

The software procedures which accomplish the processing shown in the blocks of Figure 3-53 are as follows:

[2] - GGTRVAL

[4] - GGVALDATA

[6-9] - GGVALBODY, GGVALFILE

[10] - GGVALBODY

[11] - GGTRVAL

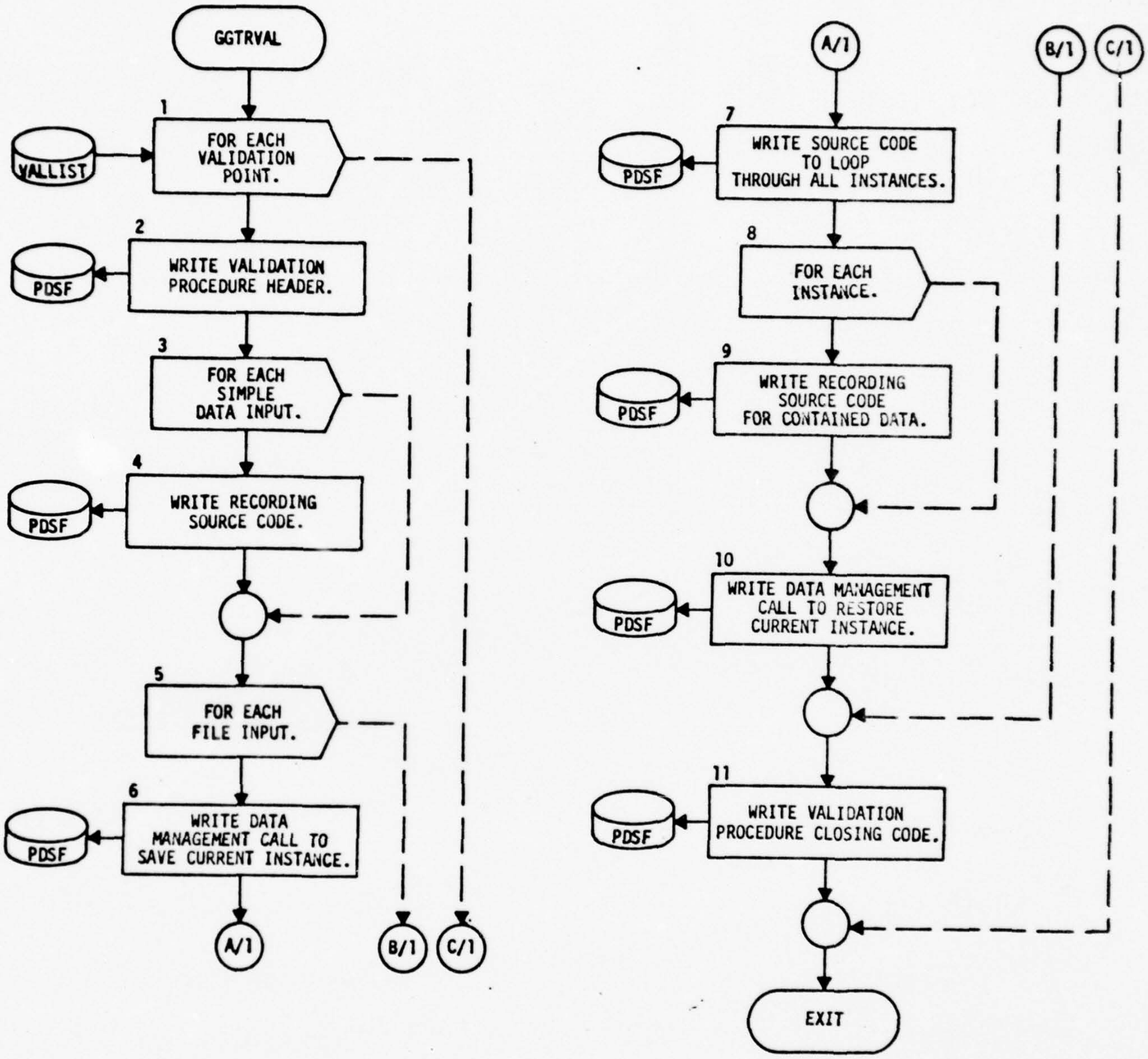



Figure 3-53 Validation Translation (GGTRVAL)

### 3.5.4 Alpha Translation (GGTRALFA)

Description

For each ALPHA in ALFALIST, the Alpha Translation module retrieves the executable description (either BETA or GAMMA) from the ASSM and translates it into ALPHA procedure source code, which is written on the Procedure Declaration Source File (PDSF). The executable description statements are expressed in either the base language (PDL 2) or in RSL. The base language statements are written unchanged to the PDSF. RSL statements, identified by scanning the BETA or GAMMA text for RSL keywords, are translated into the base language statements necessary to accomplish the specified RSL operation. The translated base language statements consist primarily of a call or sequence of calls to the various Data Management procedures. The RSL keywords recognized by GGTRALFA are CREATE, DESTROY, SELECT, FOR, and ENDFOREACH.

The CREATE, DESTROY, SELECT and FOR EACH operations that appear in BETA (GAMMA) text may legally apply only to FILEs; they may not be applied to ENTITY_CLASSes or to ENTITY_TYPEs. When GGTRALFA encounters these operations during translation, the operand is checked to ensure that it is a FILE. If not, an error diagnostic is issued via XXREVSOUT.

During the translation of an ALPHA's BETA (GAMMA) text, GGTRALFA checks for consistency with the requirements that have been specified for that ALPHA. The text is checked to determine if the INPUTS and OUTPUTS relationships that have been specified for the ALPHA are actually implemented. GGTRALFA establishes keywords for each DATA name that is the object of an INPUTS or OUTPUTS relationship having the current ALPHA as subject. During translation, GGTRALFA searches for and consistency-checks the use of declared I/O DATA names. Statements in the BETA or GAMMA text that are not consistent with specified I/O requirements result in error messages to the user, but are translated nevertheless.

In addition to translating BETA (GAMMA) text, GGTRALFA also produces source code to implement all CREATES, DESTROYS, SETS AND FORMS relationships that have been specified. The source code corresponding to CREATES and SETS relationships is inserted at the beginning of ALPHA procedures, just

after the initial 'BEGIN'. The source code corresponding to DESTROYS and FORMS relationships is inserted at the end of ALPHA procedures, just prior to the final'END;'.

Input

ASSM

| | | |
|---|---|---|
| ALPHA LIST (ALFALIST) | - | Linked list of ASSM pointers to ALPHAs that are to be included in the Simulator Program, with sublists of ALPHA relationship pointers and relationship object pointers. ALFALIST is a sublist of the Simulator Translation List (See Figures 3-40, 3-41). |

Output

| | | |
|---|---|---|
| PROCEDURE DECLARATION SOURCE FILE (PDSF) | - | A PDL 2 text file containing ALPHA procedure source code. |

Local Data

| | | |
|---|---|---|
| KEYARRAY | - | An array of pointers, each pointing to the first record in a linked list of keyword records. Each list contains records for non-PDL 2 keywords whose length equals the value of the associated KEYARRAY index. For example, KEYARRAY [5] points to the first record in the list of records for keywords of length 5. The keyword record structure is shown in Figure 3-54. |
| PDL2ARRAY | - | An array of pointers, each pointing to the first in a linked list of PDL 2 keywords. Each list contains PDL 2 keywords whose length equals the value of the associated PDL2ARRAY index. For example, PDL2ARRAY [5] points to the first in the list of PDL 2 keywords of length 5. |
| SAVARRAY | - | An array of records containing information saved during translation of an RSL FOR EACH statement for use at the end of the statement. SAVARRAY provides for nested FOR EACH statements. The index of SAVARRAY corresponds to the nesting level of the statement. For example, SAVARRAY [3] corresponds to a FOR EACH statement nested at the 3rd |

level down. The SAVARRAY records contain the operand of the statement and a local variable name that is a required parameter in a Data Manager procedure call generated when the end of the statement (ENDFOREACH) is encountered.

SAVARRAY Record

Local Variable Name
Operand Name
Operand Length

CRITARRAY - An array of records containing information saved during translation of an RSL FOR EACH statement for use at the end of the statement. CRITARRAY corresponds to the nesting level of the statement. For example, CRITARRAY [3] corresponds to a FOR EACH statement nested at the 3rd level down. The CRITARRAY records contain the information necessary for retrieval of the criterion portion of a FOR EACH statement when the end of the statement is reached.

CRITARRAY Record

Text Line Containing Beginning of Criterion
Length of Line
Pointer to First Word of Criterion in Line
ASSM Pointer to Next Segment of Text

Processing

Processing performed in the Alpha Translation module is shown in the flow diagram of Figure 3-55. The following comments refer to processing steps in the flow diagram.

[2, 8, 9] - The content of keyword records is defined in Figure 3-54.

[21, 25, 26] - The syntax of a CREATE statement is:

$$\text{CREATE operand RECORD} \left\{ \begin{matrix} ; \\ \text{END} \\ \text{ELSE} \end{matrix} \right\}_1^1$$

[22] - A CREATE operation in BETA(GAMMA) text may legally apply only to FILEs; it may not be applied to an ENTITY_CLASS.

[28, 32, 33] - The syntax of a DESTROY statement is:

$$\text{DESTROY operand RECORD} \left\{ \begin{matrix} ; \\ \text{END} \\ \text{ELSE} \end{matrix} \right\}_1^1$$

[29] - A DESTROY operation in BETA(GAMMA) text may legally apply only to FILEs; it may not be applied to an ENTITY_CLASS.

[35, 37, 43, 44] - The syntax of a SELECT statement is:

$$\text{SELECT} \left\{ \begin{matrix} \text{FIRST} \\ \text{NEXT} \end{matrix} \right\}_1^1 \text{RECORD FROM}$$

$$\text{operand [SUCH THAT (criterion)]}$$

$$\left\{ \begin{matrix} ; \\ \text{END} \\ \text{ELSE} \end{matrix} \right\}_1^1$$

[40] - A SELECT operation in BETA(GAMMA) text may legally apply only to FILEs; it may not be applied to an ENTITY_CLASS or to an ENTITY_TYPE.

[42] - If the statement is a "SELECT FIRST...", the generated source code is:

```
BEGIN
  EE8UPDATE(operand);
  EE8FIRST(operand);
```

If the statement is a "SELECT NEXT...", the generated source code is:

```
BEGIN
  EE8UPDATE(operand);
  EE8NEXT(operand);
```

[43] - A SELECT statement may be terminated with a ';', 'END' or 'ELSE' following the operand; the criterion portion of the statement is optional.

[47, 50, 51, 56] - The following source code is generated if the statement includes a criterion:

```
IF(NOT EE8ENDS(operand)) THEN
  WHILE(NOT(criterion)) AND
  (NOT EE8ENDS(operand))
  DO EE8NEXT(operand);
```

[49, 52, 53, 54] - If a variable in the criterion is an input/output keyword, then its use is tested for consistency with INPUTS and OUTPUTS relationships specified for this ALPHA. If consistent, the occurrence of the variable is recorded by incrementing the Incidence Counter in the keyword record; if not consistent, the user is notified via a message through XXREVSOUT. Keyword use is interpreted as output if followed by ':='; otherwise, it is interpreted as input.

[58] - The occurrence of 'RECORD_FOUND' as an ALPHA output is recorded by creating a keyword record for 'RECORD_FOUND' and incrementing the Incidence Counter.

[59] - If the SELECT statement is terminated by ';' or 'ELSE', then this is immediately written on the PDSF. If it is terminated by 'END', then the processing shown in steps 100 through 102 is required and is enabled by setting NEEDWORD:=FALSE.

[61, 64, 71, 73, 74] - The syntax of a FOR EACH statement is:

FOR EACH operand RECORD
[SUCH THAT (criterion)] DO

$$\text{user-code ENDFOREACH} \left\{ \begin{array}{l} ; \\ \text{END} \\ \text{ELSE} \end{array} \right\}_1^1$$

[61, 62] - 'FOR' is both a PDL 2 word and an RSL word. When 'FOR' is encountered, GGTRALFA tests the succeeding word to determine if 'FOR' is to be interpreted as RSL or PDL 2. If the succeeding word is 'EACH', then 'FOR' is interpreted as the beginning of an RSL FOR EACH statement.

[65] - A FOR EACH operation in BETA(GAMMA) text may legally apply only to FILEs; it may not be applied to an ENTITY_CLASS or to an ENTITY_TYPE.

[68] - Operand name and length are stored in SAVARRAY.

[69] - The generated source code is:

```
BEGIN
  EE8UPDATE(operand);
  EE8FIRST(operand);.
```

[77] - Information necessary to retrieve the criterion is stored in CRITARRAY.

[78, 81, 82, 86] - The following source code is generated if the statement includes a criterion:

```
IF(NOT EE8ENDS(operand)) THEN
  WHILE(NOT EE8ENDS(operand))
  AND(NOT(criterion)) DO
  EE8NEXT(operand);
```

[80, 83, 84, 85] - If a variable in the criterion is an input/output keyword, then its use is tested for consistency with INPUTS and OUTPUTS relationships specified for this ALPHA. If consistent, the occurrence of the variable is recorded by incrementing the Incidence Counter in the keyword record; if not consistent, the user is notified via a message through XXREVSOUT. Keyword use is interpreted as output if followed by ':='; otherwise, it is interpreted as input.

[87] - The following source code is generated:

```
RECORD_FOUND:=EE8FOUND(operand);
WHILE RECORD_FOUND DO
BEGIN
  EE8SAVDS(operand, local variable);
```

The local variable is obtained from SAVARRAY.

[88] - The occurrence of 'RECORD_FOUND' as an ALPHA output is recorded by creating a keyword record for 'RECORD_FOUND' and incrementing the Incidence_Counter.

[89, 96] - The level counter FORECHNUM identifies the nesting level of the FOR EACH statement being processed. For example, FORECHNUM=3 corresponds to a FOR EACH statement nested at the third level down.

[90] - The generated source code is:

```
EE8UPDATE(operand);
EE8RESDS(operand, local variable);
EE8NEXT(operand);
```

The operand and local variable are obtained from SAVARRAY.

[92, 93, 94] - The following source code is generated if the statement includes a criterion:

```
IF(NOT EE8ENDS(operand))
THEN WHILE (NOT EE8ENDS(operand))
AND (NOT(criterion)) DO
EE8NEXT(operand);
```

The operand is retrieved from SAVARRAY; information necessary to retrieve the criterion is obtained from CRITARRAY.

[99] - For SETS relationships, the generated source code is:

```
EE8SETYP(operand);
```

For CREATES relationships, the generated code is:

```
EE8CREATE(operand);
```

[101] - For FORMS relationships the generated source code is:

```
EE8FORM(operand);
```

For DESTROYS relationships the generated source code is:

```
EE8DESTROY(operand);
```

[104] - Keyword use is interpreted as output if following by ':='; otherwise, it is interpreted as input.

[105] - Consistency is tested by comparing keyword use with the keyword type that is listed in the keyword record.

[107] - The Incidence Counter of the keyword record is incremented.

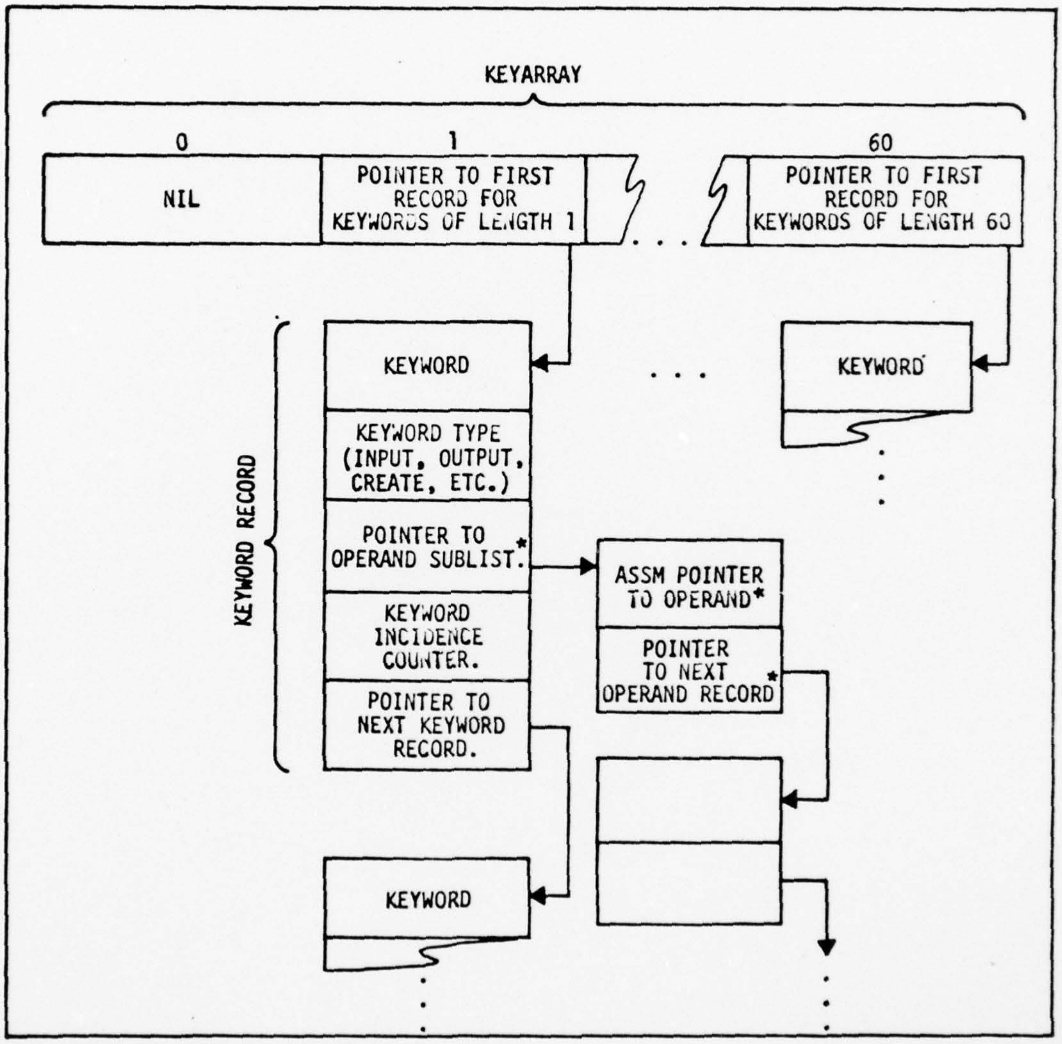
| | | |
|---|---|---|
| [109-111] | - | Input/Output consistency is tested by examining the Incidence Counter values in each of the input/output keyword records. For each Incidence Counter that is equal to zero, the user is notified via XXREVSOUT that there is an INPUTS or OUTPUTS relationship that is not implemented in the BETA (or GAMMA) text. |

Procedure References

The following list correlates the functional processing steps shown in Figure 3-55 with the REVS procedures in which the processing is performed.

| | | |
|---|---|---|
| [1] | - | GGALFREL |
| [2] | - | GGKEYPDL |
| [8] | - | GGKEYWRD |
| [9] | - | GGKEYWRD |
| [13] | - | GGLINGEN |
| [17] | - | GGWRDGEN |
| [19] | - | GGCOMPAR, GGTRRSL |
| [21 - 27] | - | GGCREATE |
| [21] | - | GGGETOPR |
| [22] | - | GGOPRCHK |
| [25] | - | GGGETOPR |
| [26] | - | GGCOMPAR |
| [28 - 34] | - | GGDSTROY |
| [28] | - | GGGETOPR |
| [29] | - | GGOPRCHK |
| [32] | - | GGGETOPR |
| [33] | - | GGCOMPAR |
| [35 - 59] | - | GGSELECT |
| [35, 37] | - | GGGETOPR |
| [36] | - | GGCOMPAR |
| [40] | - | GGOPRCHK |
| [43] | - | GGENDSELECT |
| [44] | - | GGGETOPR |
| [45] | - | GGCOMPAR |
| [48] | - | GGLINGEN, GGWRDGEN, GGCOMTST |

| | | |
|---|---|---|
| [49] | - | GGCOMPAR |
| [51 - 54] | - | GGINOUT |
| [55] | - | GGENDSELECT |
| [58] | - | GGFOUND |
| [60 - 88] | - | GGFORECH |
| [60] | - | GGGETOPR |
| [61] | - | GGCOMPAR |
| [63] | - | GGGETOPR |
| [65] | - | GGOPRCHK |
| [70, 74] | - | GGGETOPR |
| [71, 73, 75] | - | GGCOMPAR |
| [79] | - | GGLINGEN, GGWRDGEN, GGCOMTST |
| [80] | - | GGCOMPAR |
| [82 - 85] | - | GGINOUT |
| [88] | - | GGFOUND |
| [89 - 96] | - | GGENDECH |
| [93] | - | GGLINGEN, GGWRDGEN, GGCOMTST |
| [97 - 102] | - | GGPDL2 |
| [99, 101] | - | GGDATAMGMT |
| [103 - 107] | - | GGINOUT |
| [108] | - | GGCOMTST |
| [109 - 111] | - | GGRSLTST |
| [112] | - | GGDISPOS |




*NOT USED IN CURRENT VERSION OF GGTRALFA.

Figure 3-54 Keyword Record Structure

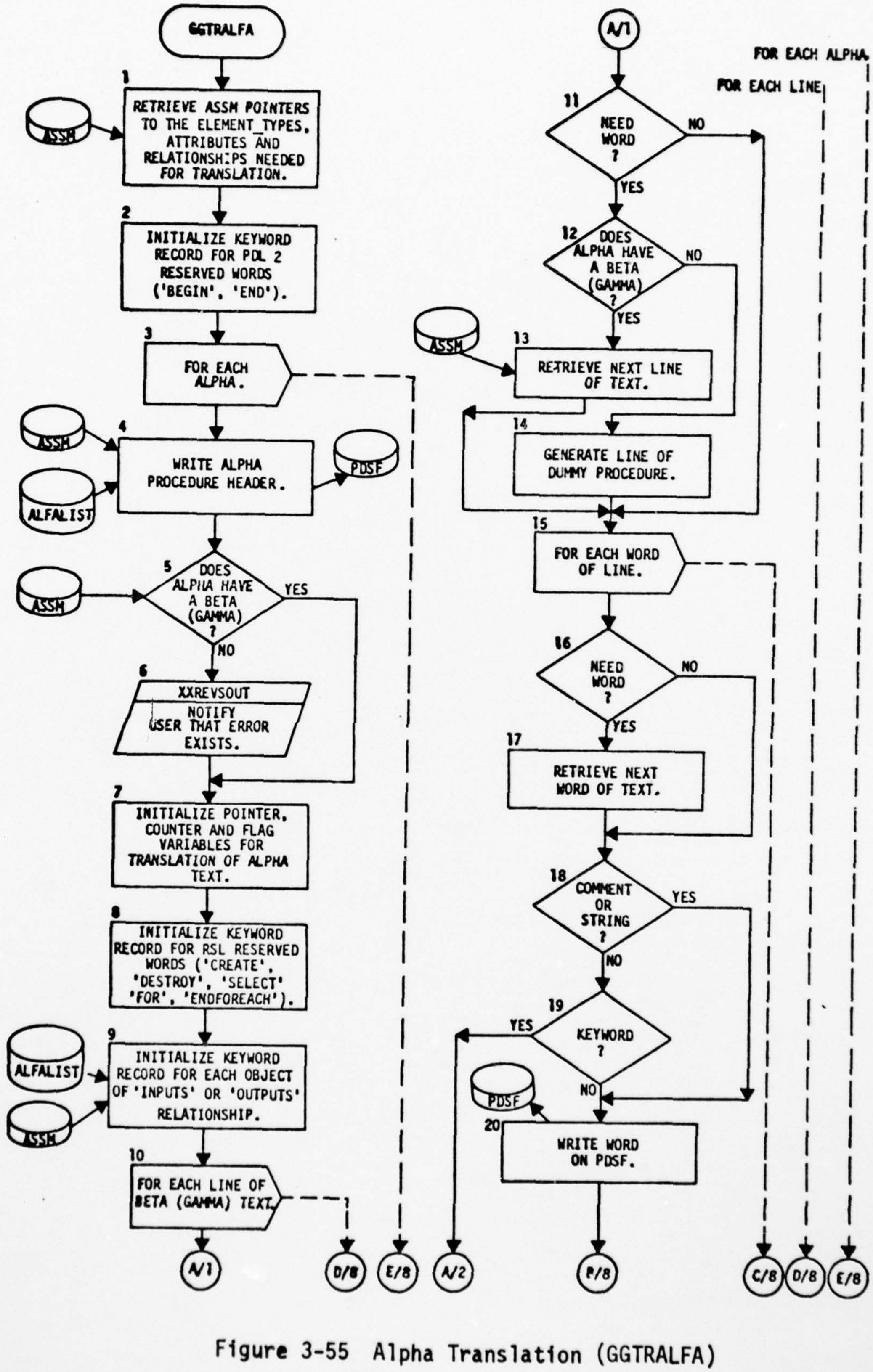



Figure 3-55 Alpha Translation (GGTRALFA)

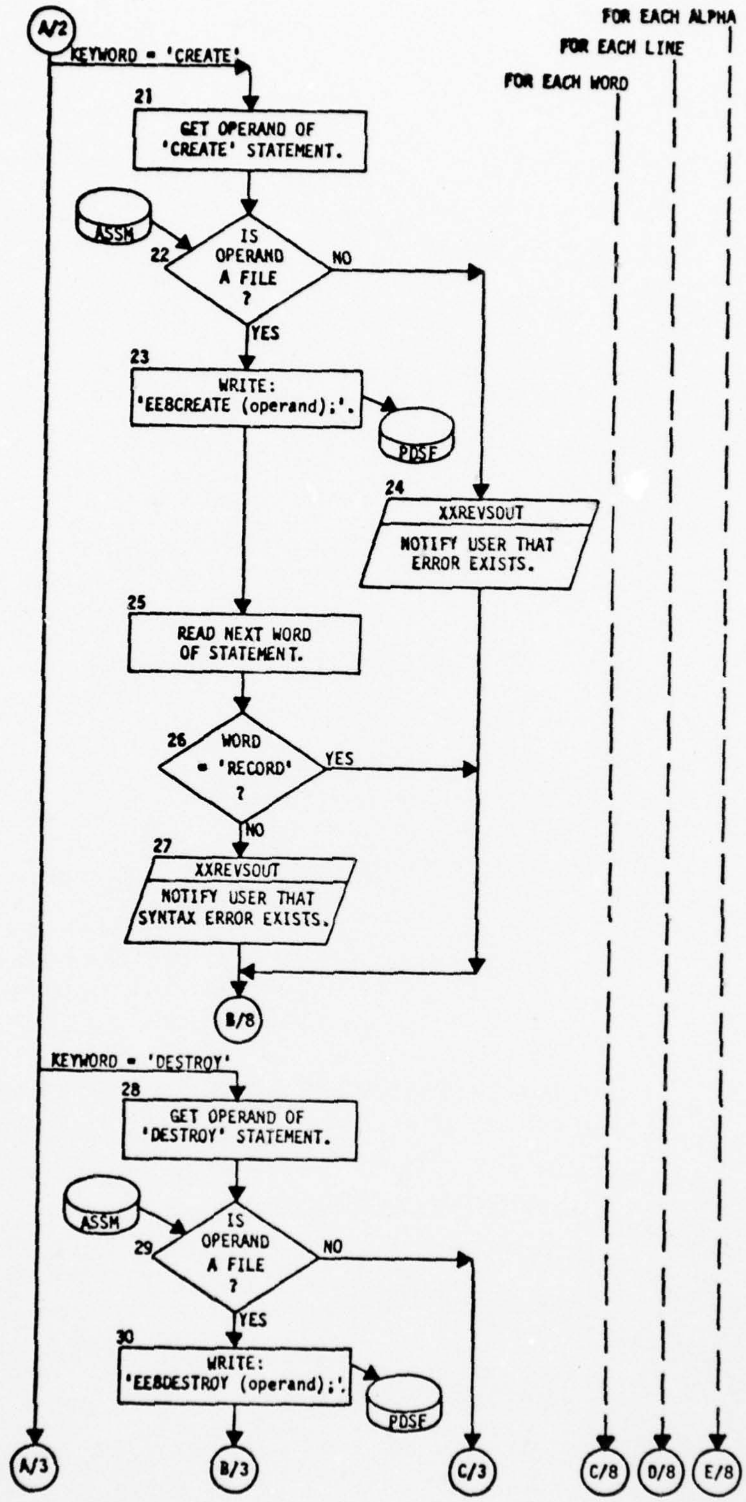



Figure 3-55 Alpha Translation (GGTRALFA) (Continued)

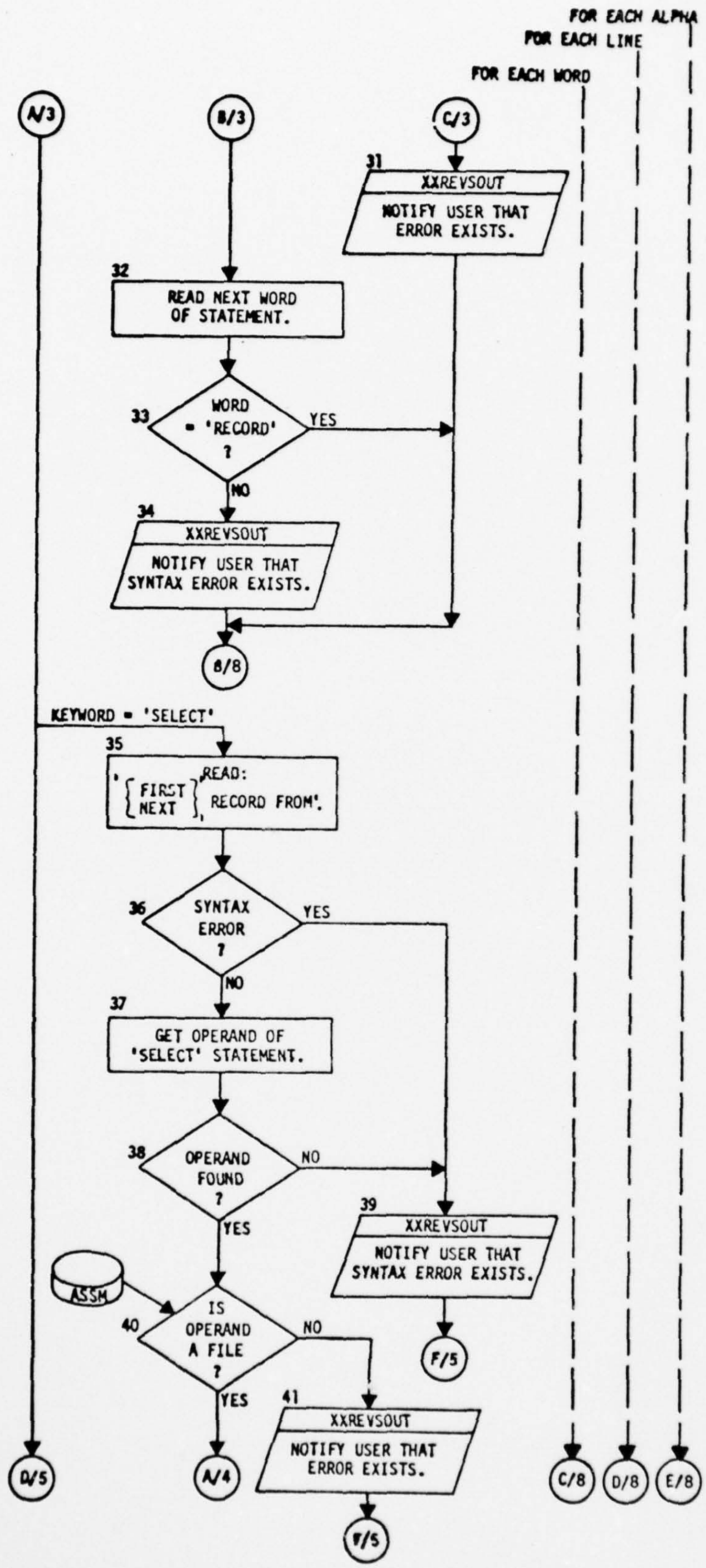



Figure 3-55 Alpha Translation (GGTRALFA) (Continued)

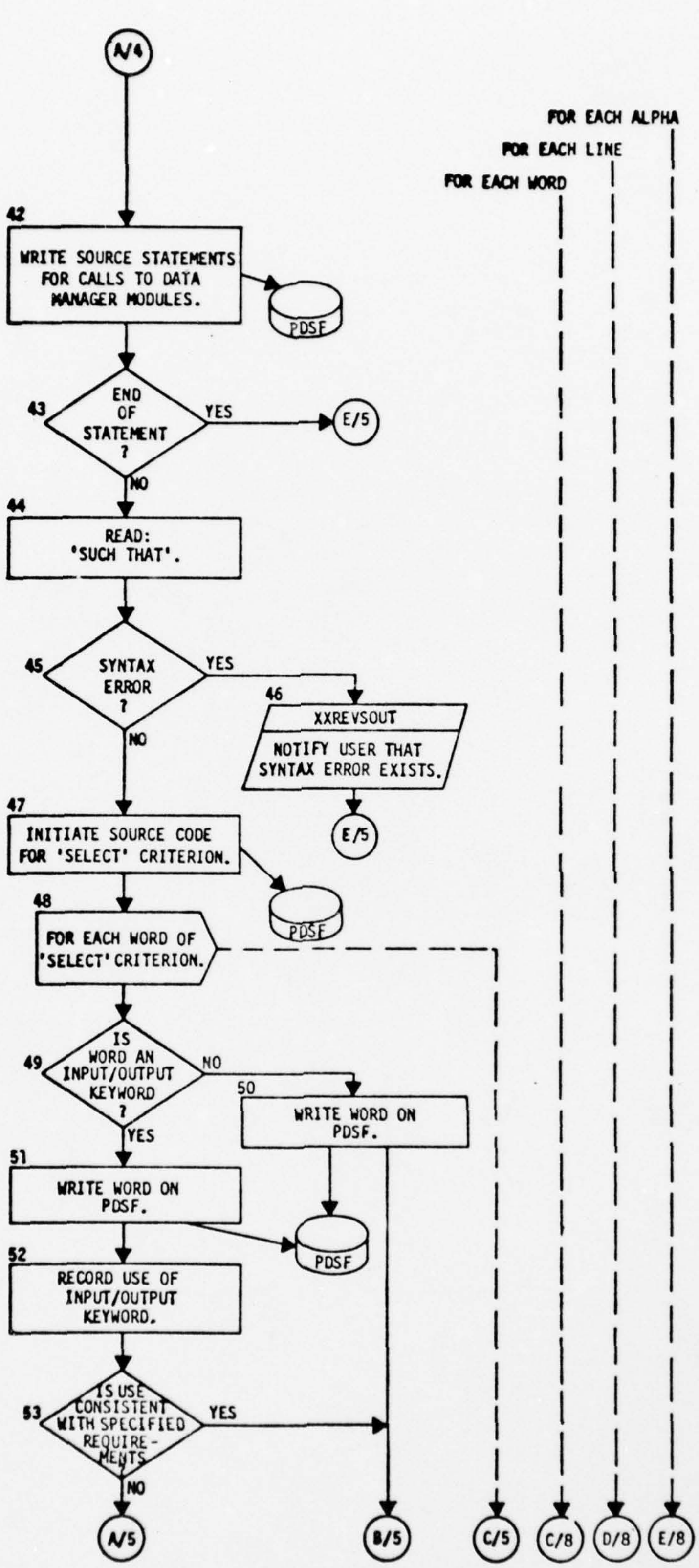



Figure 3-55 Alpha Translation (GGTRALFA) (Continued)

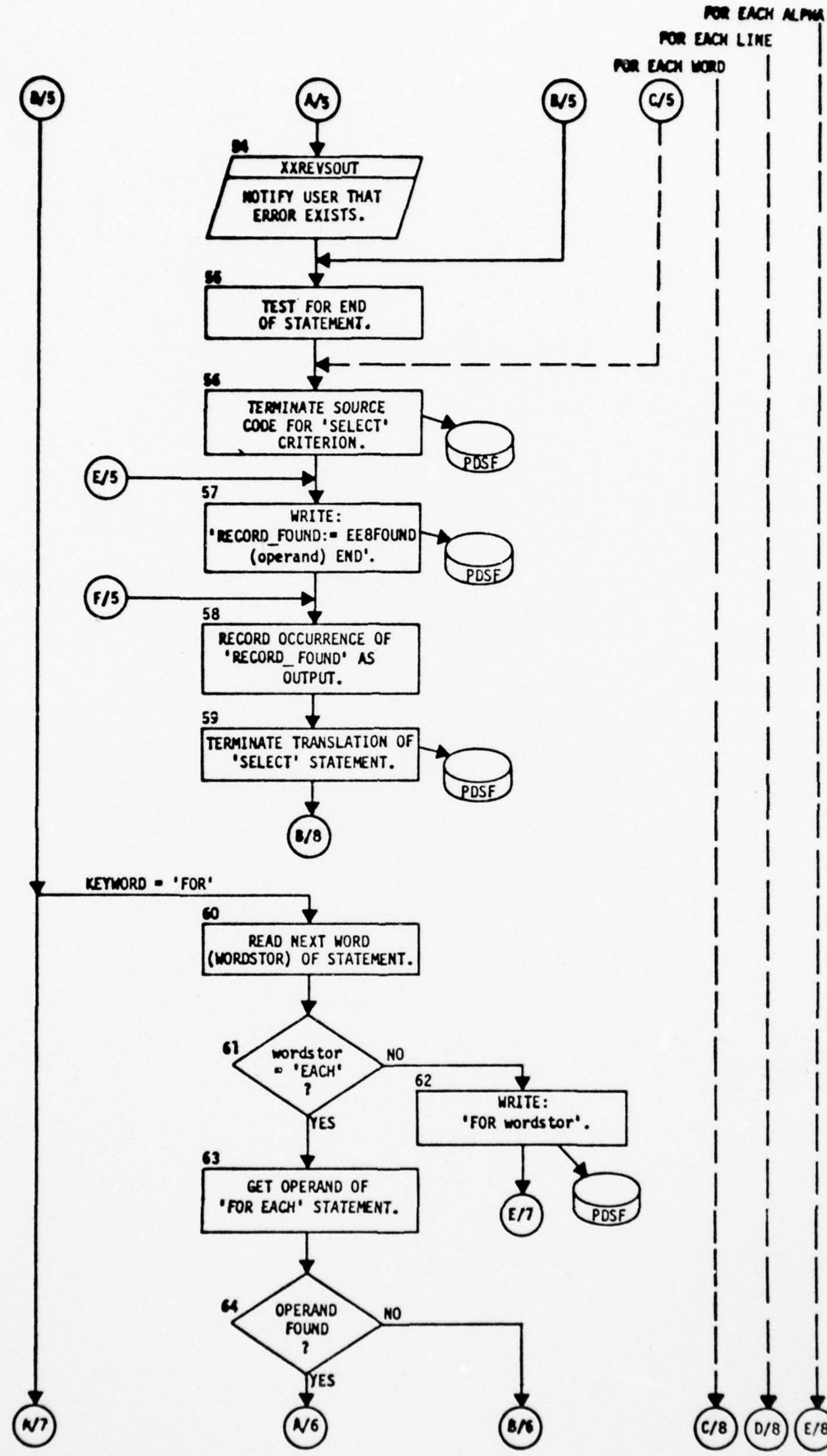



Figure 3-55 Alpha Translation (GGTRALFA) (Continued)

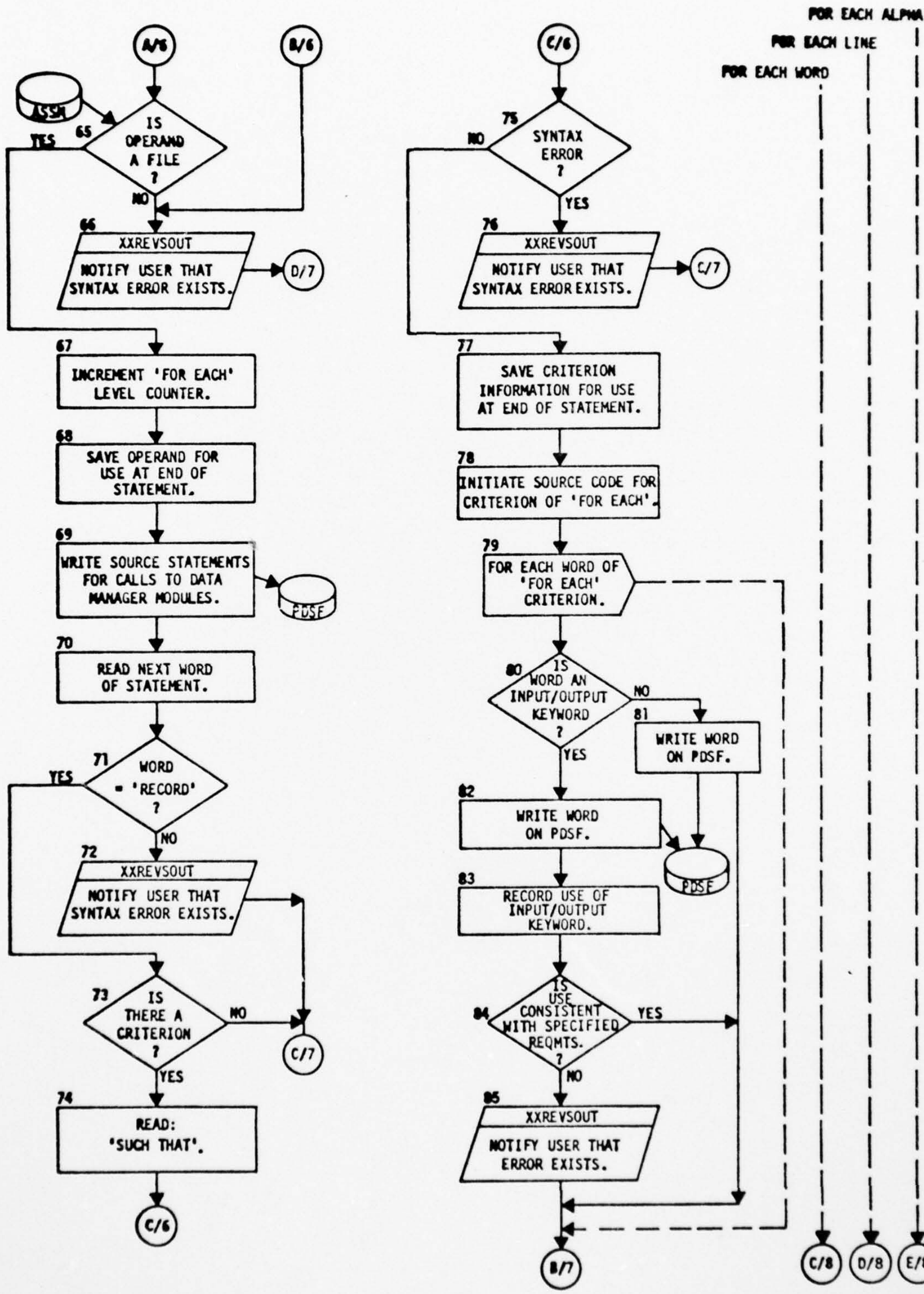



Figure 3-55 Alpha Translation (GGTRALFA) (Continued)

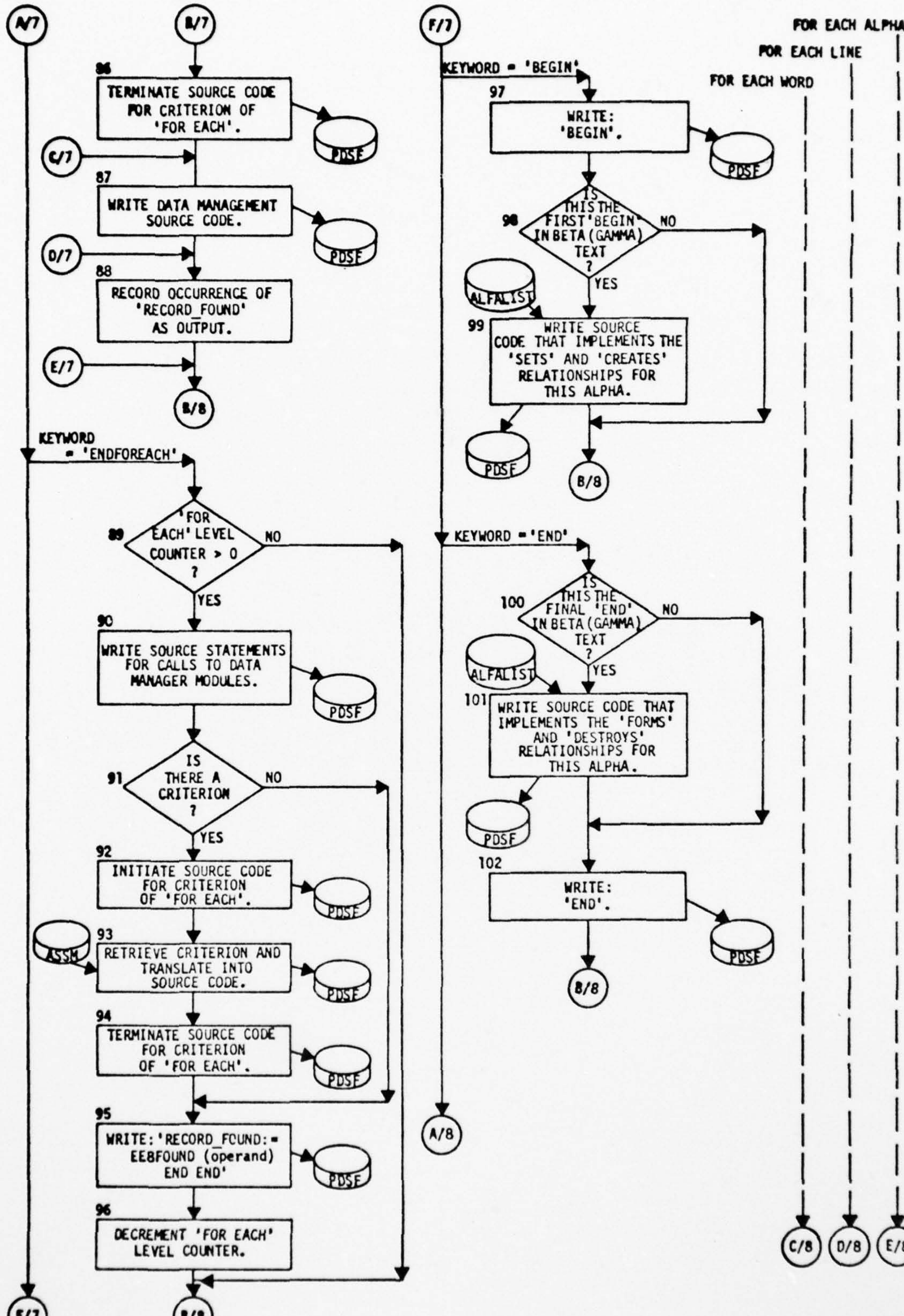



Figure 3-55 Alpha Translation (GGTRALFA) (Continued)

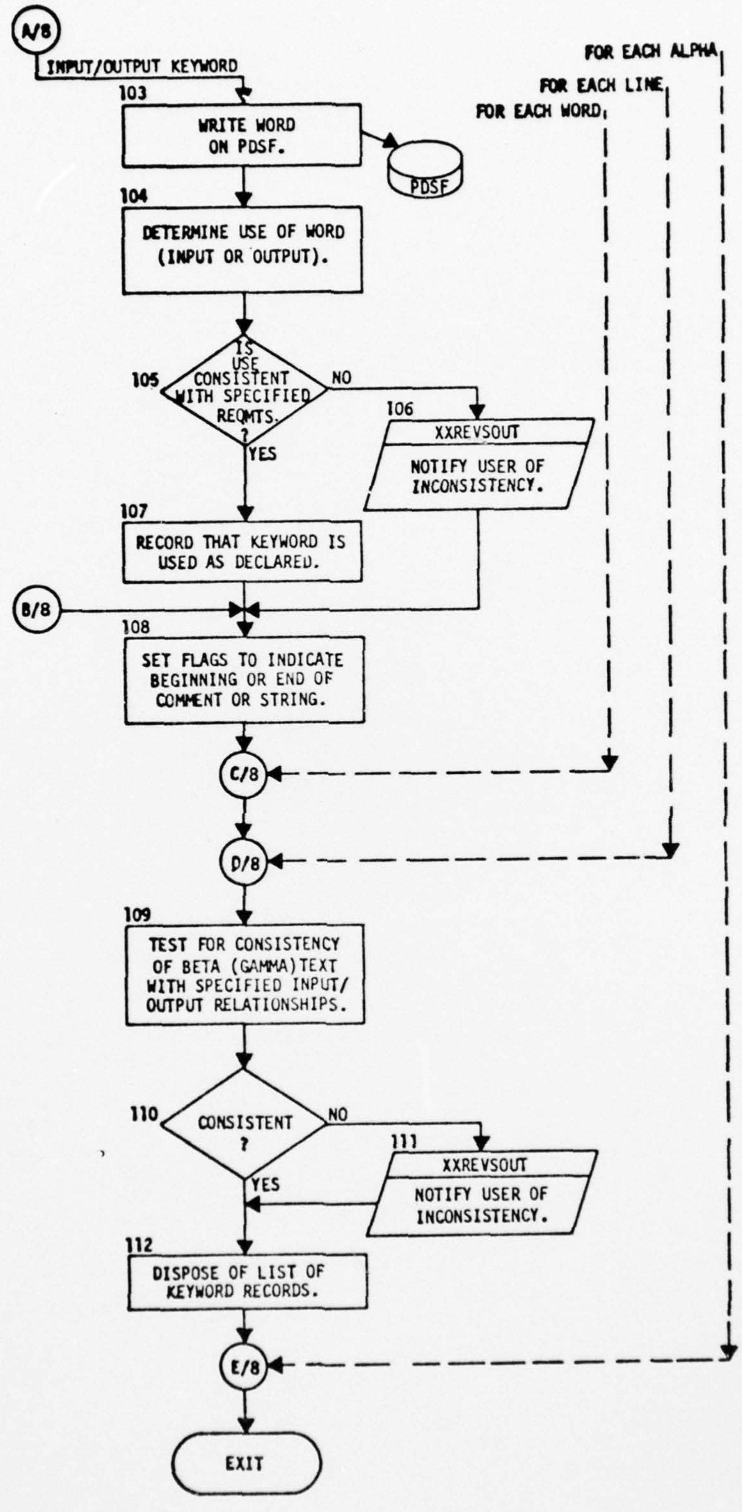



Figure 3-55 Alpha Translation (GGTRALFA) (Continued)

### 3.5.5 R-Net/Subnet Translation (GGTRRNET)

Description

R-Net/Subnet Translation generates executable R_NET and SUBNET source code procedures for consolidation into the Simulator Program. The R-Net/Subnet Translation module obtains retrieval information for those R_NETs and SUBNETs to be translated from RNETLIST and SNETLIST. Structural information for these R_NETs and SUBNETs is obtained from the ASSM. This structural information includes node types, element types, element names, successor-predecessor relationships among nodes, and association relationships between elements and nodes. The only additional source of information required by R-Net/Subnet Translation is the Event Definition File, EEDF. The EEDF provides an event number parameter which is assigned to each EVENT, INPUT_INTERFACE and OUTPUT_INTERFACE by Event/Enablement Translation.

These sources provide all the requirements dependent information necessary for R-Net/Subnet Translation to generate executable source code R_NET and SUBNET procedures in the base language. The SUBNET and R_NET procedures are written on the Procedures Declaration Source File (PDSF). Consolidation inserts the procedures into the Simulator Program by copying from this file to the Compile File (CF).

Input

| | | |
|---|---|---|
| ASSM | - | R_NET and SUBNET names, structures and relationships. |
| R_NET LIST (RNETLIST) | - | List of R_NETs to be included in Simulator Program. |
| SUBNET LIST (SNETLIST) | - | List of SUBNETs to be included in Simulator Program. |
| EVENT/ENABLEMENT DEFINITION FILE (EEDF) | - | File that contains an Event Definition Record for every EVENT, INPUT_INTERFACE, and OUPTUT_INTERFACE referenced by the requirements model. The event number parameter is used in R-Net/Subnet Translation. The content of an event Definition Record is defined in Section 3.5. |

### Output

| | | |
|---|---|---|
| PROCEDURE DECLARATION SOURCE FILE (PDSF) | - | File containing executable SUBNET and R_NET source code procedures for those SUBNETs and R_NETs referenced by the requirements model. |

### Local Files

| | | |
|---|---|---|
| COMPILE FILE (CF) | - | Scratch file used to contain main body code for the procedure being generated. |
| VARIABLES FILE (VF) | - | File used to contain variable declarations for the procedure being generated. |

### Processing

Processing is shown by the functional flow diagram of Figure 3-56. The following comments refer to processing steps in the flow diagram.

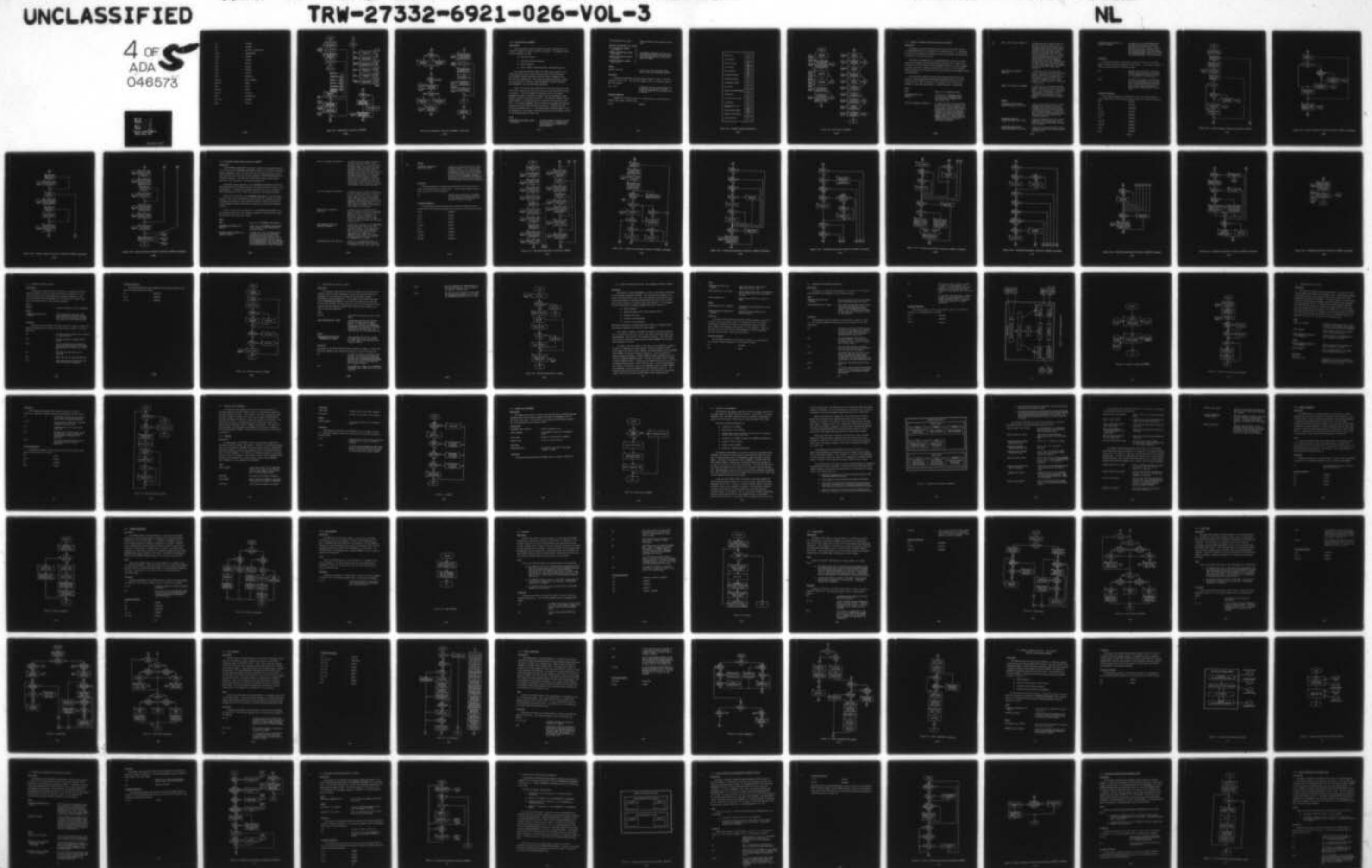
| | | |
|---|---|---|
| [3-20] | - | A label is allocated for the main branch of the R_NET or SUBNET. Labels are required at branch ends. Execution is to be transferred to the branch end when a termination within a SUBNET is encountered along the branch. |
| [8-34] | - | The main branch label is written at the procedure end. |
| [17] | - | If there are no ordinals, the RSL ordering of the OR branches is maintained in the generated code. |
| [18] | - | A label is allocated for each OR branch and written at the branch end. |
| [22] | - | A label is allocated for each AND branch. It is written at the branch end. |

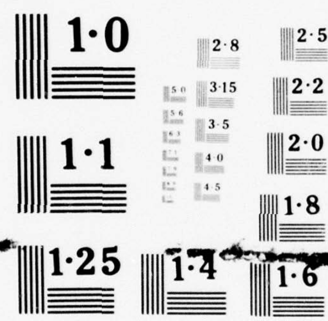
### Procedure References

In the flow diagram of Figure 3-56, the processing steps correspond to code in GGTRRNET procedures as follows:

AD-A046 573 TRW DEFENSE AND SPACE SYSTEMS GROUP HUNTSVILLE ALA F/G 9/2
REVS MAINTENANCE MANUAL. SREP FINAL REPORT. VOLUME III.(U)
AUG 77 W E BENOIT, P N BERGSTRESSER DASG60-75-C-0022
UNCLASSIFIED TRW-27332-6921-026-VOL-3 NL

4 OF 5
ADA
046573






NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

| | | |
|---|---|---|
| [1] | - | GGTRRNET |
| [2] | - | GGTRRNET |
| [3] | - | GGTRRNET, GGETNEXTRNET |
| [4-6] | - | GGBRANCHPROS |
| [7,8] | - | GGMERGE |
| [10] | - | GGALPHA |
| [11] | - | GGINPUT |
| [12] | - | GGVALID |
| [13] | - | GGSUBNET |
| [14] | - | GGEVENT |
| [15] | - | GGOUTFACE |
| [16-19] | - | GGOR, GGCNSOR |
| [20-22] | - | GGAND |
| [23] | - | GGFOR |
| [24-28] | - | GGFOR |
| [29-31] | - | GGTRSUBN |
| [32] | - | GGBRANCHPROS |
| [33, 34] | - | GGSMERGE |
| [35] | - | GGSELCT |

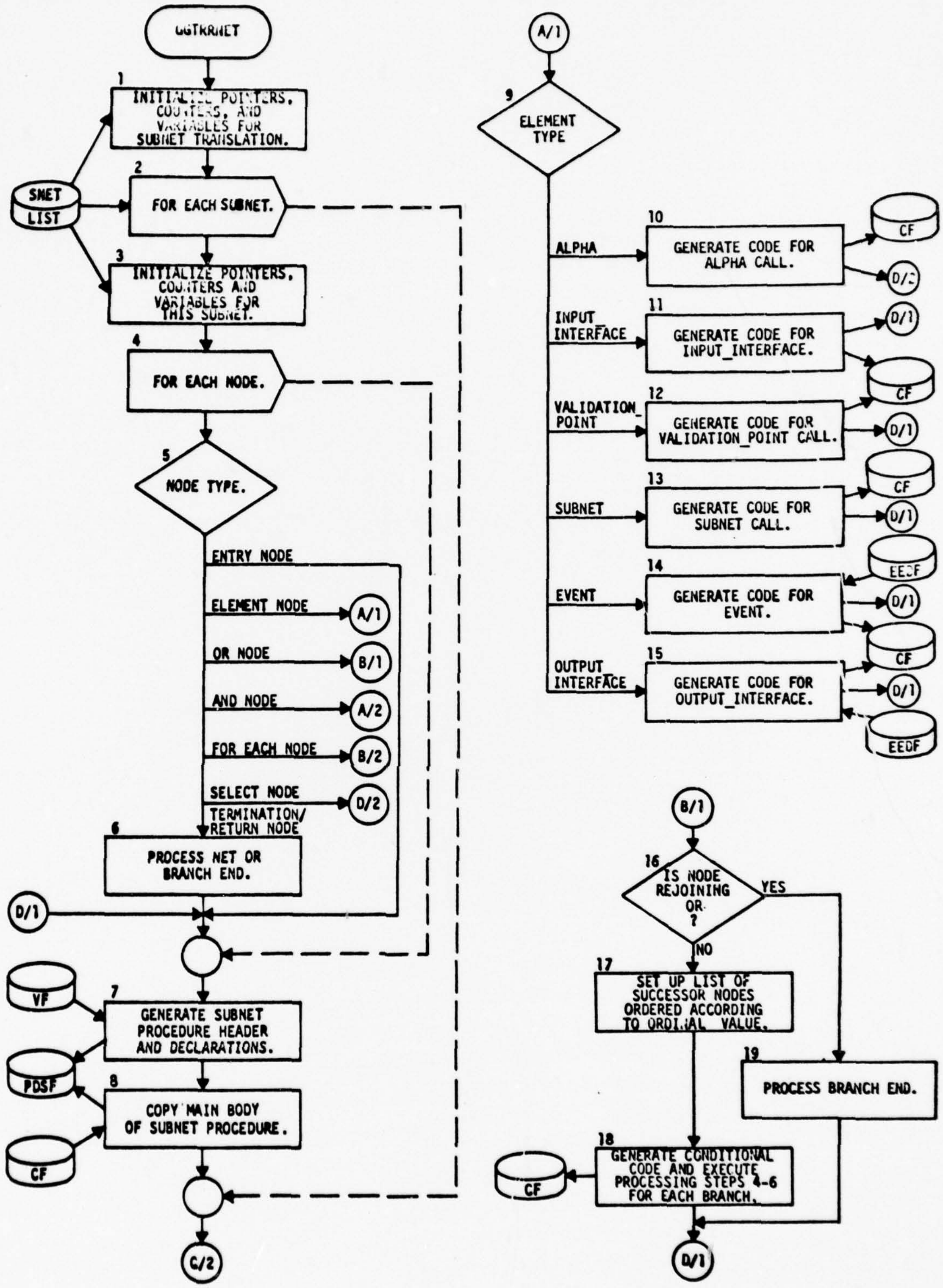



Figure 3-56 R-Net/Subnet Translation (GGTRRNET)

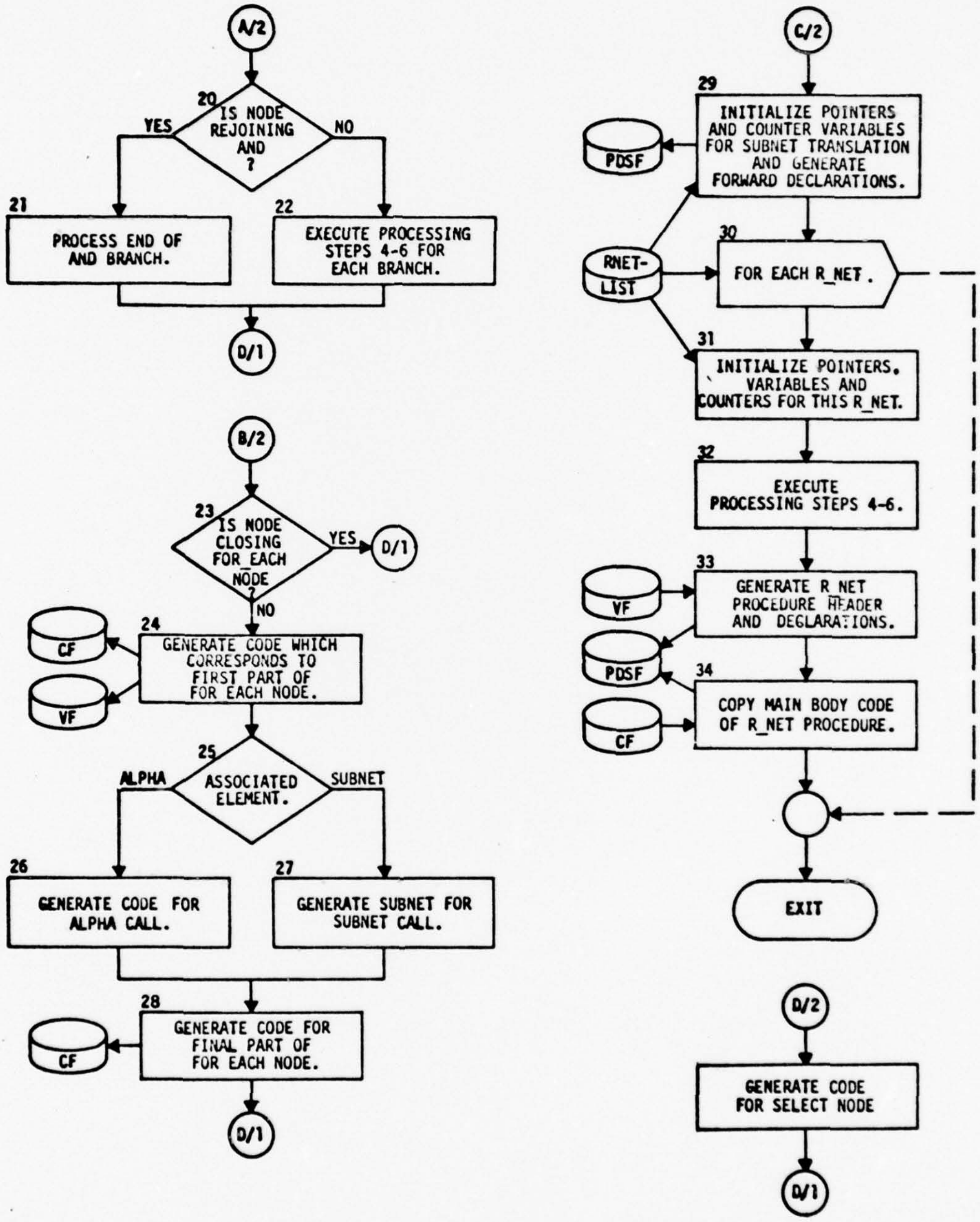



Figure 3-56 R-Net/Subnet Translation (GGTRRNET) (Continued)

### 3.5.6 Consolidation (GGCONSOL)

#### Description

The Consolidation module assembles the source components of the Simulator Program for input to the PDL 2 compiler. The components that must be assembled include:

- Data Declarations
- Simulation Support Procedures
- SETS Procedures
- R_NET, SUBNET, VALIDATION_POINT, and ALPHA Procedures

The Consolidation module merges the source code produced by the SIMGEN translator modules with the source code for SETS procedures and simulation support procedures which are input to SIMGEN via the SDF and RISF, respectively. The simulation support procedures include the Initialization, Executive, Event Management, Procedure Scheduler, Data Management, and Data Recording procedures.

Figure 3-57 defines the organization of the source code to be sent to the PDL 2 compiler and shows the input text files from which the code is obtained. The Data Management procedures include requirements dependent procedures contained on the PDSF file and requirements independent procedures contained on the RISF. The Procedure Scheduler procedure (EESCHED) is contained partially on the RISF and partially on the PSSF. As indicated, GGCONSOL inserts the PSSF text between the beginning and end texts for EESCHED obtained from the RISF. Because each item (variable, procedure, etc.) in the program must be defined before it can be referenced in the program, the source code sent to the compiler must be in the order depicted in Figure 3-57.

#### Input

| | |
|---|---|
| REQUIREMENTS INDEPENDENT SOURCE - FILE (RISF) | The RISF contains Simulator Program source code that is independent of the particular requirements model being generated. |

SETS DEFINITION FILE (SDF) - The SDF contains SETS procedure source code.

CONSTANT DECLARATIONS FILE (CDF)
TYPE DECLARATIONS SOURCE FILE (TDSF)
VARIABLE DECLARATIONS SOURCE FILE (VDSF)
PROCEDURE DECLARATION SOURCE FILE (PDSF)
PROCEDURE SCHEDULER SOURCE FILE (PSSF)

Requirements dependent text files containing the Simulator Program source code generated during the translation phases of SIMGEN.

Output

COMPILE FILE (CF) - A PDL 2 text file containing source code for the entire Simulator Program.

Processing

Processing performed by the Consolidation module is shown in the flow diagram of Figure 3-58. The following comments refer to processing steps in the flow diagram.

[2-4, 6-8] - Information from the various input files is copied to the Compile File until either the character $ or an end-of-file is reached.

Procedure Reference

GGCONSOL calls REVS procedures as indicated below to perform the processing steps shown in Figure 3-58.

[2-16] - GGCOPIER

| | |
|---|---|
| PROGRAM HEADER | RISF |
| LABEL DECLARATIONS | RISF |
| CONSTANT DECLARATIONS | RISF<br>SDF<br>CDF |
| TYPE DECLARATIONS | RISF<br>TDSF<br>SDF |
| VARIABLE DECLARATIONS | RISF<br>SDF<br>VDSF |
| DATA MANAGEMENT PROCEDURES | PDSF<br>RISF |
| DATA RECORDING PROCEDURES | RISF |
| EVENT MANAGEMENT PROCEDURES | RISF |
| SETS PROCEDURES | SDF |
| SIMULATOR INITIALIZATION PROCEDURES | RISF |
| ALPHA PROCEDURES | PDSF |
| VALIDATION AND SUBNET PROCEDURES | PDSF |
| R_NET PROCEDURES | PDSF |
| SIMULATOR EXECUTIVE UTILITY PROCEDURES | RISF |
| PROCEDURE SCHEDULER PROCEDURE | RISF<br>PSSF<br>RISF |
| SIMULATOR EXECUTIVE PROCEDURE | RISF |
| SIMULATOR PROGRAM BODY | RISF |

Figure 3-57 Simulator Program Organization

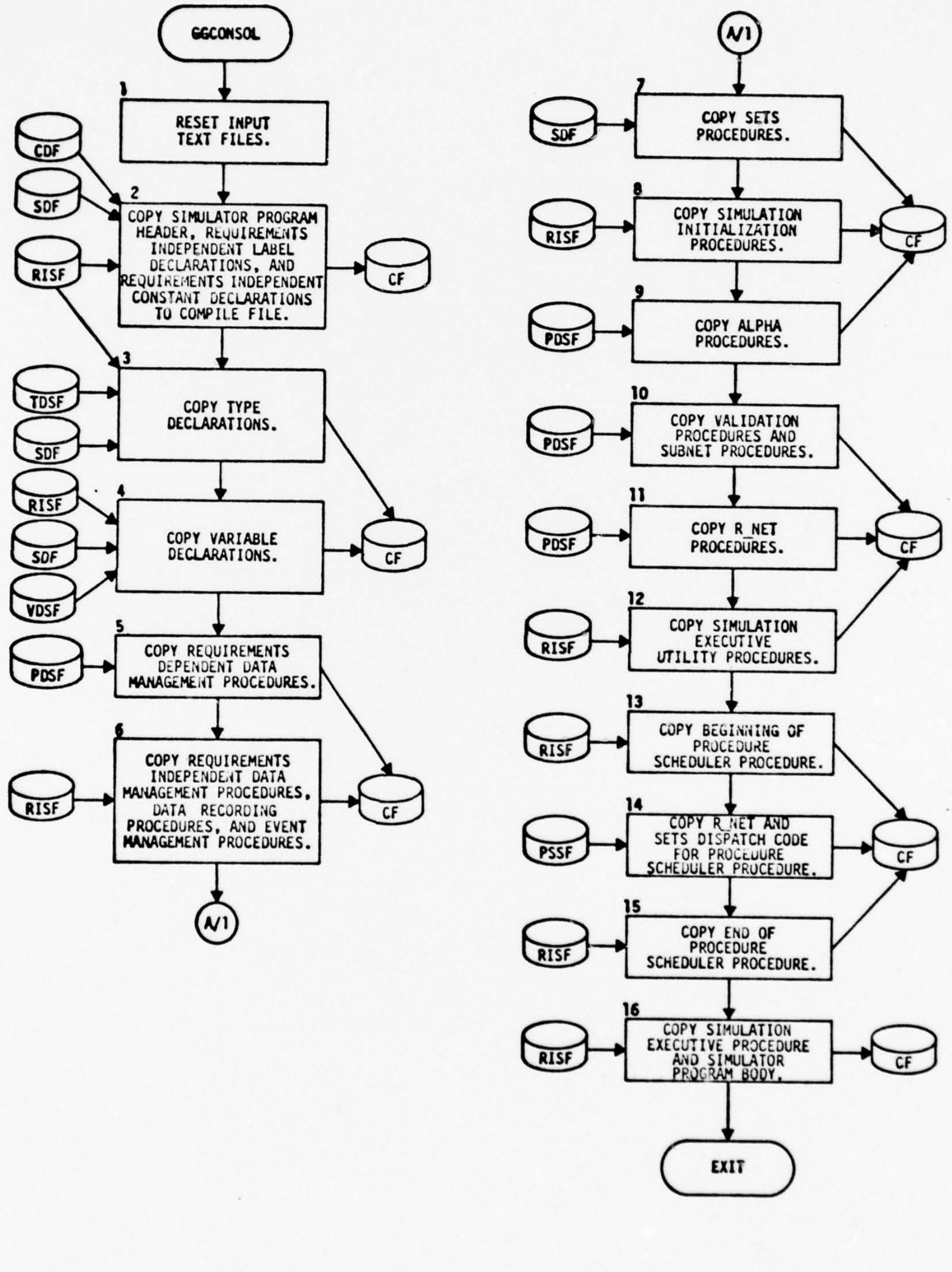



Figure 3-58 Consolidation (GGCONSOL)

### 3.5.7 Analytic Simulator Validation Translation (GGTRVP)

Description

The Analytic Simulator Validation Translation module (ASVT) analyzes all VALIDATION_POINTs to be included in the simulator build. It determines the data to be collected at simulator run time and generates the source code to implement recording of the data.

The source code generated for a VALIDATION_POINT consists of calls to owner-check procedures which guarantee that owned data has a proper owner selected, of calls to Simulator Data Management routines to access instances of FILEs, and of PDL 2 output procedure calls to write to the VALIDATION_POINT recording file the contents of FILEs and DATA which are RECORDed by the VALIDATION_POINT.

The generated recording and owner-check procedure declarations are written to the VALIDATION_POINT PROCEDURE DECLARATION SOURCE FILE (PDSF). During R_NET and SUBNET translation, an invocation of the appropriate procedure is then generated whenever a VALIDATION_POINT is referenced.

Input

| | | |
|---|---|---|
| ASSM | - | Names of all VALIDATION_POINTs. |
| VALIDATION_POINT LIST (VALLIST) | - | Linked list of VALIDATION_POINTs with sublists of RECORDed simple and FILE DATA and RECORDed FILEs (See Figure 3-47). |
| FILE LIST (headed by GG9LFILST) | - | A linked list of all FILEs processed by the Data Translation module. Each entry of the list has a sub-list of all the FILE's owners (ENTITY_CLASS or ENTITY_TYPEs), the FILE's name (an AASTRING), the FILE's ASSM address, a LOCALITY flag (BOOLEAN) indicating whether the FILE is LOCAL (FALSE) or GLOBAL (TRUE), and an owner-check ID (INTEGER) identifying the FILE's owner-check procedure. |

| | | |
|---|---|---|
| DATA LIST (headed by GG9DECLST) | - | A linked list of all DATA processed by the Data Translation module. Each entry of the list has a sub-list of all the DATA's owners (FILEs, ENTITY_CLASS, or ENTITY_TYPEs), the DATA's name (an AASTRING), the DATA's ASSM address, a pointer into the DATA_TYPE LIST indicating the DATA's PDL 2 TYPE, a pointer into the DATA_INITIAL_VALUE LIST indicating the DATA's INITIAL_VALUE, an INITIAL_VALUE status flag (INTEGER) indicating that the DATA's INITIAL_VALUE has been looked for in the ASSM (1) or not (0), and an owner-check ID (INTEGER) identifying the DATA's owner-check procedure. |
| DATA_TYPE LIST (headed by GG9TYPLST) | - | A linked list of all the DATA_TYPEs (PDL 2) processed by the Data Translation Module. Each entry of the list has the DATA_TYPE's name (an AASTRING), the DATA_TYPE's ordinal (INTEGER), and a scratch flag (BOOLEAN) used in other modules to indicate whether the DATA_TYPE is used or not. |
| OWNERS LIST (headed by GG9OWNRL) | - | A linked list of all the DATA and FILE owners processed by the Data Translation module. Each entry of the list has the owner's name (an AASTRING), the owner's ASSM address, and the owner's type (enumerated type designating no type, MESSAGE, FILE, ENTITY_TYPE, and ENTITY_CLASS). |

<u>Outputs</u>

| | | |
|---|---|---|
| VALIDATION_POINT PROCEDURE DECLARATION SOURCE FILE (PDSF) | - | A text file containing the PDL 2 source statements declaring the owner-check procedures for all DATA and FILEs processed by the Analytic Simulator Validation Translation module. In addition, the file contains the PDL 2 source statements declaring the VALIDATION_POINT recording procedures. |
| VALIDATION_POINT TYPE DECLARATION SOURCE FILE (TDSF) | - | A text file containing the PDL 2 source statements declaring the type used in declaring the VALIDATION_POINT recording file's variants. |
| VALIDATION_POINT VARIABLE DECLARATION SOURCE FILE (VDSF) | - | A text file containing the PDL 2 source statements declaring the VALIDATION_POINT recording file. |

| | | |
|---|---|---|
| VALIDATION_POINT INTERNAL LIST (headed by GG9VPL) | - | A linked list of all VALIDATION_POINTs processed by the Analytic Simulator Validation Translation Module. Each entry of the list has a sub-list of all the VALIDATION_POINT's RECORDed FILEs, the VALIDATION_POINT's name (an AASTRING), the VALIDATION_POINT's ASSM address, and the VALIDATION_POINT's ordinal (INTEGER). |

Processing

Processing performed in the Analytic Simulator Validation Translation module is shown in the flow diagram of Figure 3-58.1. The following comments refer to processing steps in the flow diagram.

| | | |
|---|---|---|
| [1] | - | Includes such operations as assigning initial values to variables and arrays for use throughout the Analytic Simulator Validation Translation module. |
| [32] | - | Includes writing out to file TDSF the PDL 2 source statements to declare the type of the VALIDATION_POINT recording file's variant and writing out to file VDSF the PDL 2 source statements to declare the VALIDATION_POINT recording file. |

Procedure References

The following correlates the functional processing elements shown in Figure 3-58.1 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | GG4VPINI |
| [2] | - | GG4VPROC |
| [3-4] | - | GG4PREAMB |
| [5-10] | - | GG4DICNT |
| [11-16] | - | GG4FLCNT |
| [17] | - | GG4VPROC |
| [18-19] | - | GG4RECSD |
| [20-30] | - | GG4RECFD |
| [31] | - | GG4VPROC |
| [32] | - | GG4VPEND |

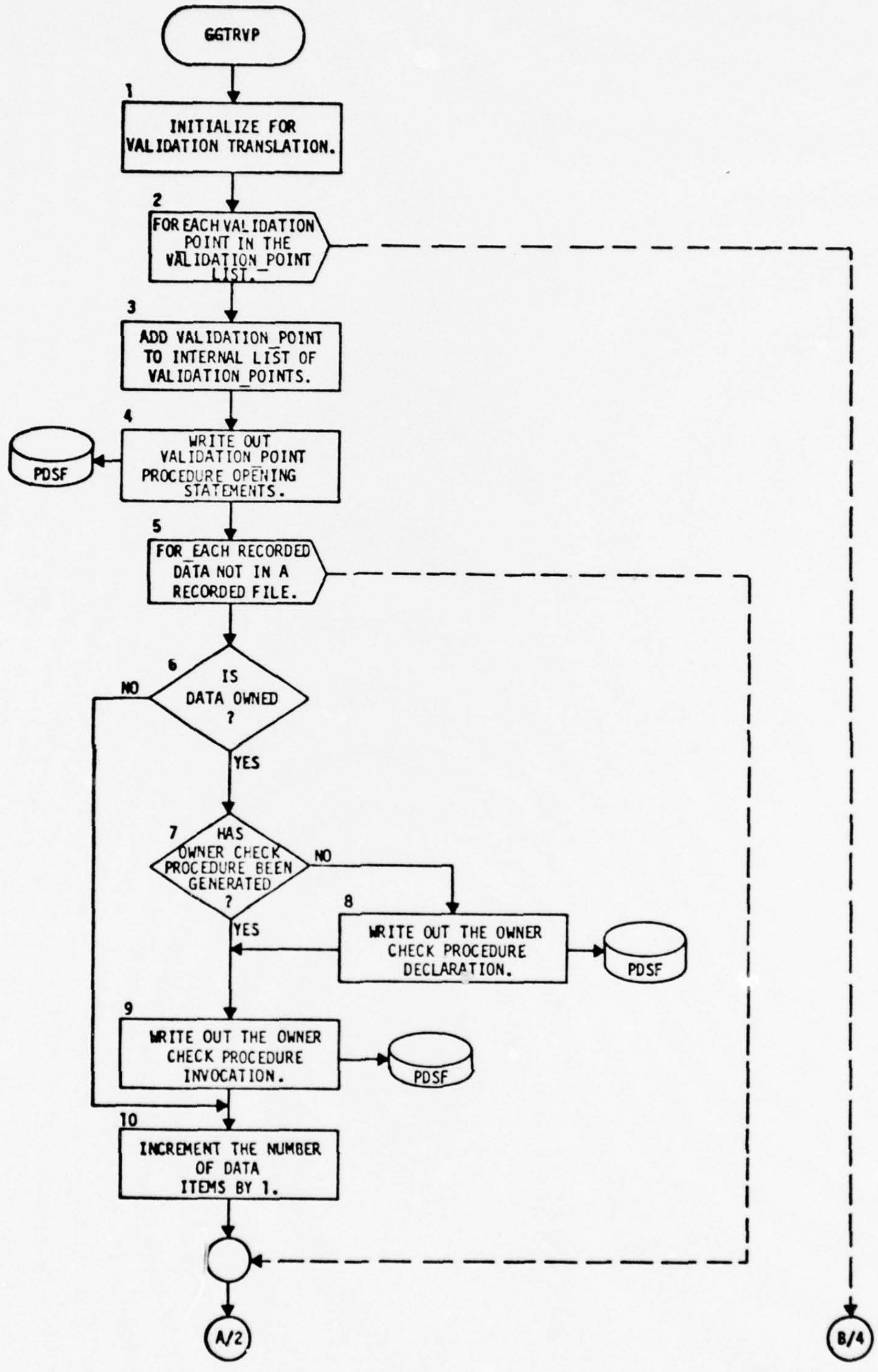



Figure 3-58.1 Analytic Simulator Validation Translation (GGTRVP)

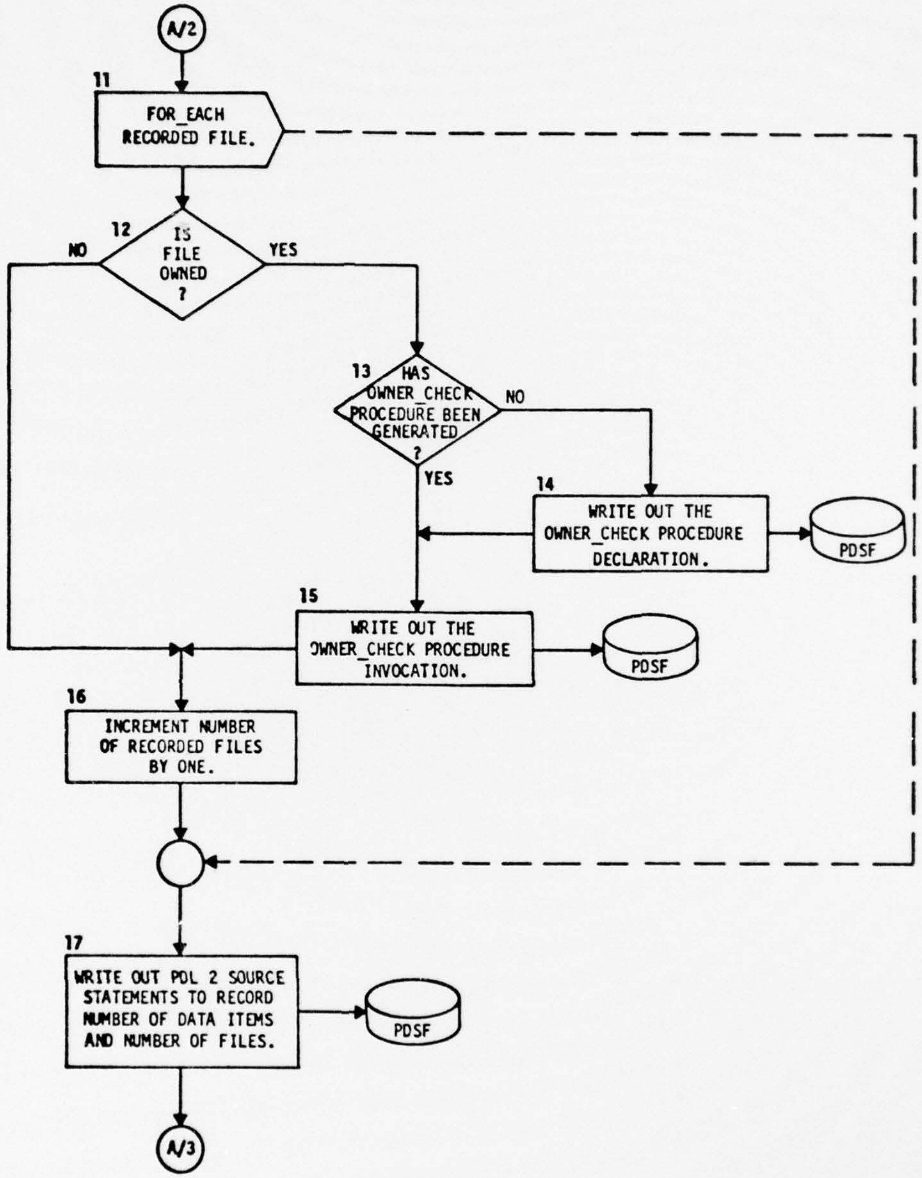



Figure 3-58.1 Analytic Simulator Validation Translation (GGTRVP) (Continued)

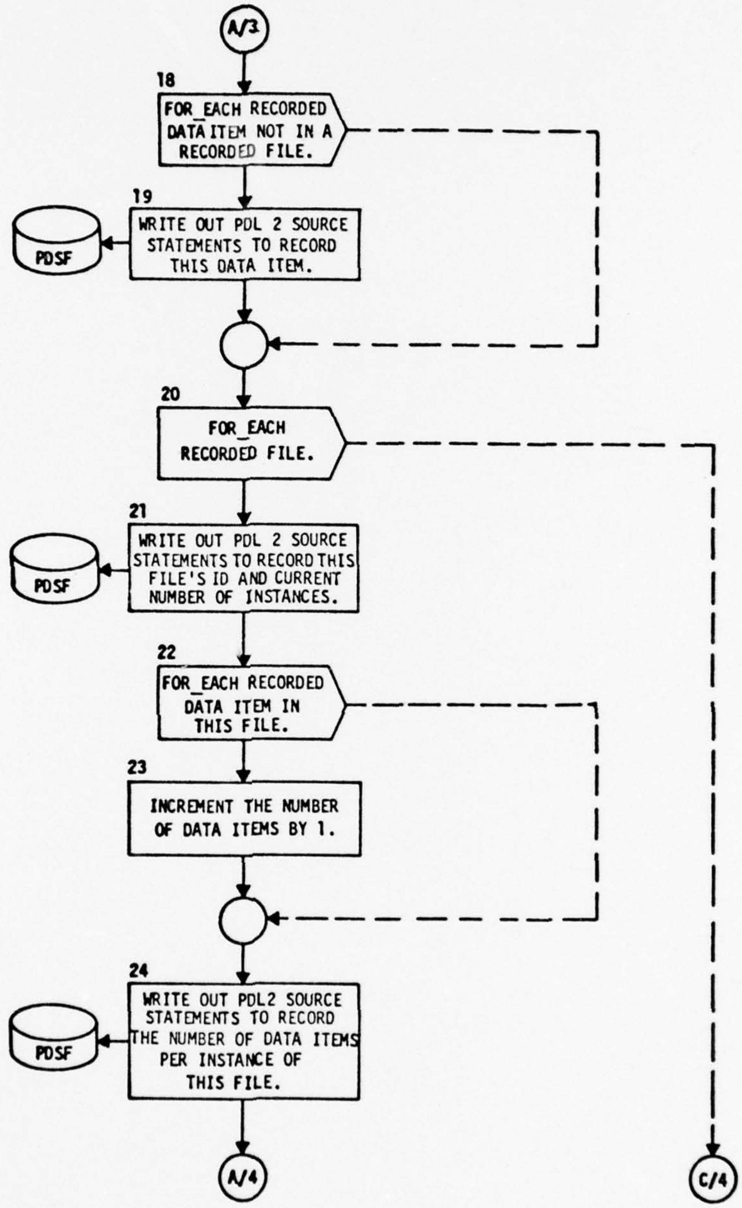



Figure 3-58.1 Analytic Simulator Validation Translation (GGTRVP) (Continued)

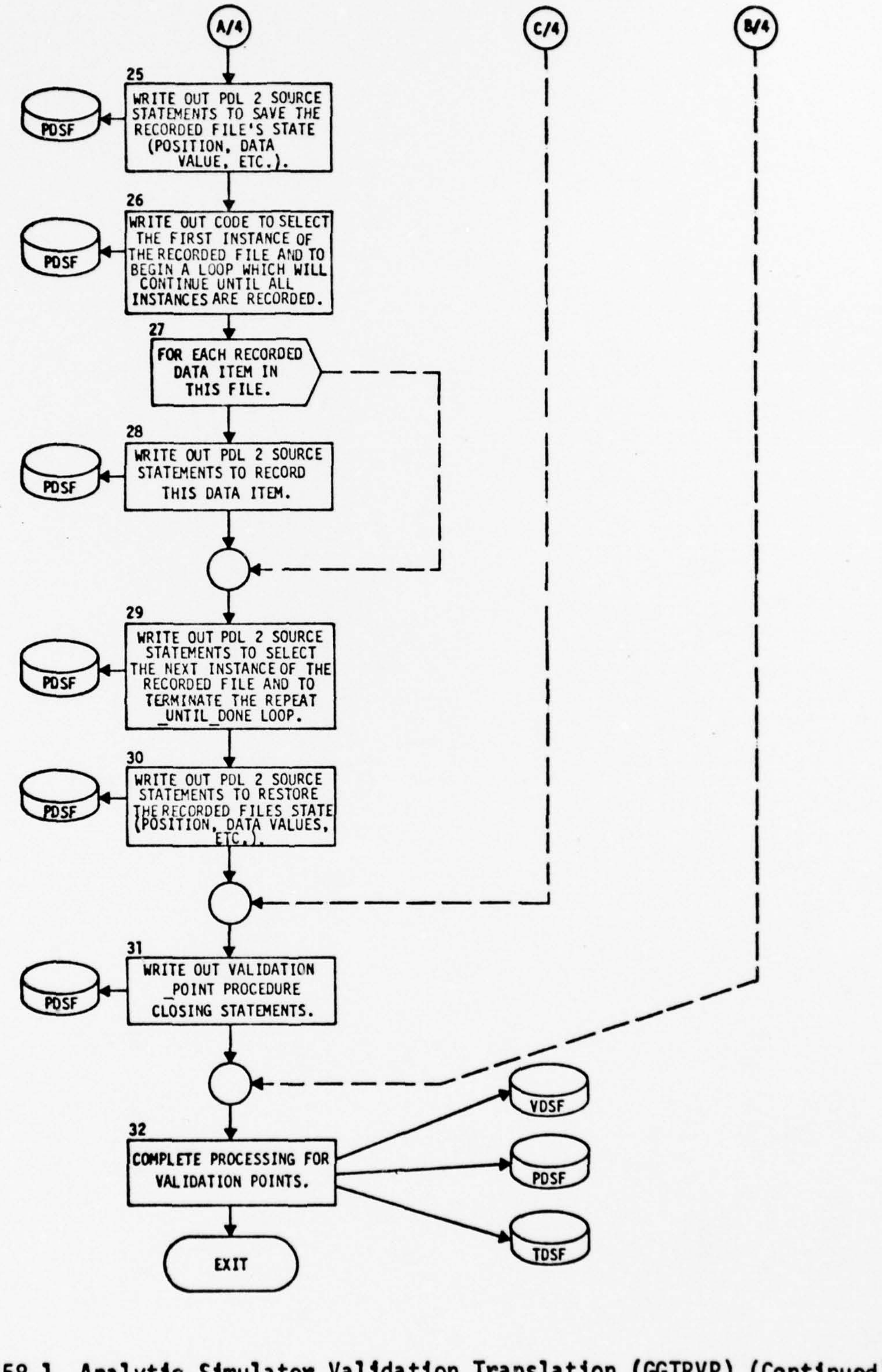



Figure 3-58.1 Analytic Simulator Validation Translation (GGTRVP) (Continued)

3.5.8 Performance-Requirement Translation (GGTRPR)

Description

The PERFORMANCE_REQUIREMENT Translation converts the executable description (TEST) of each PERFORMANCE_REQUIREMENT specified in the PRLIST into PDL 2 source statements in such a way that the user can select the test to be run and can access within the test the data recorded at VALIDATION_POINTs during simulator execution. The three basic translation steps are data manager generation, TEST translation, and executive generation.

During data manager generation, the PERFORMANCE_REQUIREMENT translation module generates the procedures and data structures which allow access to the recorded data. In general one procedure and one array entry is created for each VALIDATION_POINT and for each FILE recorded at each VALIDATION_POINT.

During TEST translation, the PERFORMANCE_REQUIREMENT translation module writes all PDL 2 source statements directly to the TEST procedure file with no change but converts all RSL statements (RETRIEVE, SELECT, and FOR EACH) to data manager procedure invocations as required to perform the requested function.

Finally, during executive generation, the PERFORMANCE_REQUIREMENT translation module constructs a main program to optionally invoke the TEST procedures and print their pass/fail results.

Input

| | | |
|---|---|---|
| ASSM | - | Names of all PERFORMANCE_REQUIREMENTs. |
| PERFORMANCE_REQUIREMENT LIST (PRLIST) | - | Linked list of PERFORMANCE_REQUIREMENTs to be included in the Simulator Post-Processor Program. |
| VALIDATION_POINT INTERNAL LIST (headed by GG9VPL) | - | A linked list of all VALIDATION_POINTs processed by the Analytic Simulator Validation Translation Module. Each entry of the list has a sub-list of all the VALIDATION_POINT's RECORDed FILEs, the VALIDATION_POINT's name (an AASTRING), the VALIDATION_POINT's ASSM address, and the VALIDATION_POINT's ordinal (INTEGER). |

| | |
|---|---|
| DATA LIST (headed by GG9DECLST) - | A linked list of all DATA processed by the Data Translation Module. Each entry of the list has a sub-list of all the DATA's owners (FILEs, ENTITY_CLASS, or ENTITY_TYPEs), the DATA's name (an AASTRING), the DATA's ASSM address, a pointer into the DATA_TYPE LIST indicating the DATA's PDL 2 TYPE, a pointer into the DATA_INITIAL_VALUE LIST indicating the DATA's INITIAL_VALUE, an INITIAL_VALUE status flag (INTEGER) indicating that the DATA's INITIAL_VALUE has been looked for in the ASSM (1) or not (0), and an owner-check ID (INTEGER) identifying the DATA's owner-check procedure. |
| FILE LIST (headed by GG9LFILST) - | A linked list of all FILEs processed by the Data Translation module. Each entry of the list has a sub-list of all the FILE's owners (ENTITY_CLASS or ENTITY_TYPEs), the FILE's name (an AASTRING), the FILE's ASSM address, a LOCALITY flag (BOOLEAN) indicating whether the FILE is LOCAL (FALSE) or GLOBAL (TRUE), and an owner-check ID (INTEGER) identifying the FILE's owner-check procedure. |
| DATA_TYPE LIST (headed by GG9TYPLST) - | A linked list of all the DATA_TYPEs (PDL 2) processed by the Data Translation module. Each entry of the list has the DATA_TYPE's name (an AASTRING), the DATA_TYPE's ordinal (INTEGER), and a scratch flag (BOOLEAN) used in other modules to indicate whether the DATA_TYPE is used or not. |
| DATA_ENUMERATED_TYPE LIST (headed by GG9ENMLST) - | A linked list defining all the ENUMERATED_TYPEs of DATA processed by the Data Translation module. Each entry of the list has a pointer to the ENUMERATED_TYPE's entry in the DATA_TYPE LIST, a sub-list of values composing the ENUMERATED TYPE's RANGE, and a counter giving the number of values composing the ENUMERATED TYPE's RANGE. |
| VALIDATION_POINT LIST (VALLIST) - | Linked list of VALIDATION_POINTs with sub-lists of RECORDed simple data and of RECORDed FILE data (see Figure 3-47). |

### Outputs

| | | |
|---|---|---|
| PERFORMANCE_REQUIREMENT SOURCE FILE (PRSF) | - | A text file containing the PDL 2 source statements declaring the Simulator Post Processor Program. This includes data declarations for DATA and FILEs RECORDED by VALIDATION_POINTs, procedure declarations for the data manager, procedure declarations for PERFORMANCE_REQUIREMENTs' executable descriptions, and a procedure for the Simulator Post Processor executive. |

### Processing

Processing performed in Performance Requirement Translation module is shown in the flow diagram of Figure 3-58.2. The following comments refer to processing steps in the flow diagram.

| | | |
|---|---|---|
| [1] | - | Includes such operations as assigning initial values to variables and arrays for use throughout the Performance Requirement Translation module. |

### Procedure References

The following correlates the functional processing elements shown in Figure 3-58.2 with the REVS procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1-5] | - | GG5PRBGN |
| [6-8] | - | GG5VVPV |
| [9-11] | - | GG5VVSD |
| [12] | - | GG5VVPV |
| [13-18] | - | GG5VVFL |
| [19] | - | GG5VVPV |
| [20-22] | - | GG5PRPR |
| [23-96] | - | GG5PRTST |
| [97-98] | - | GG5PREND |

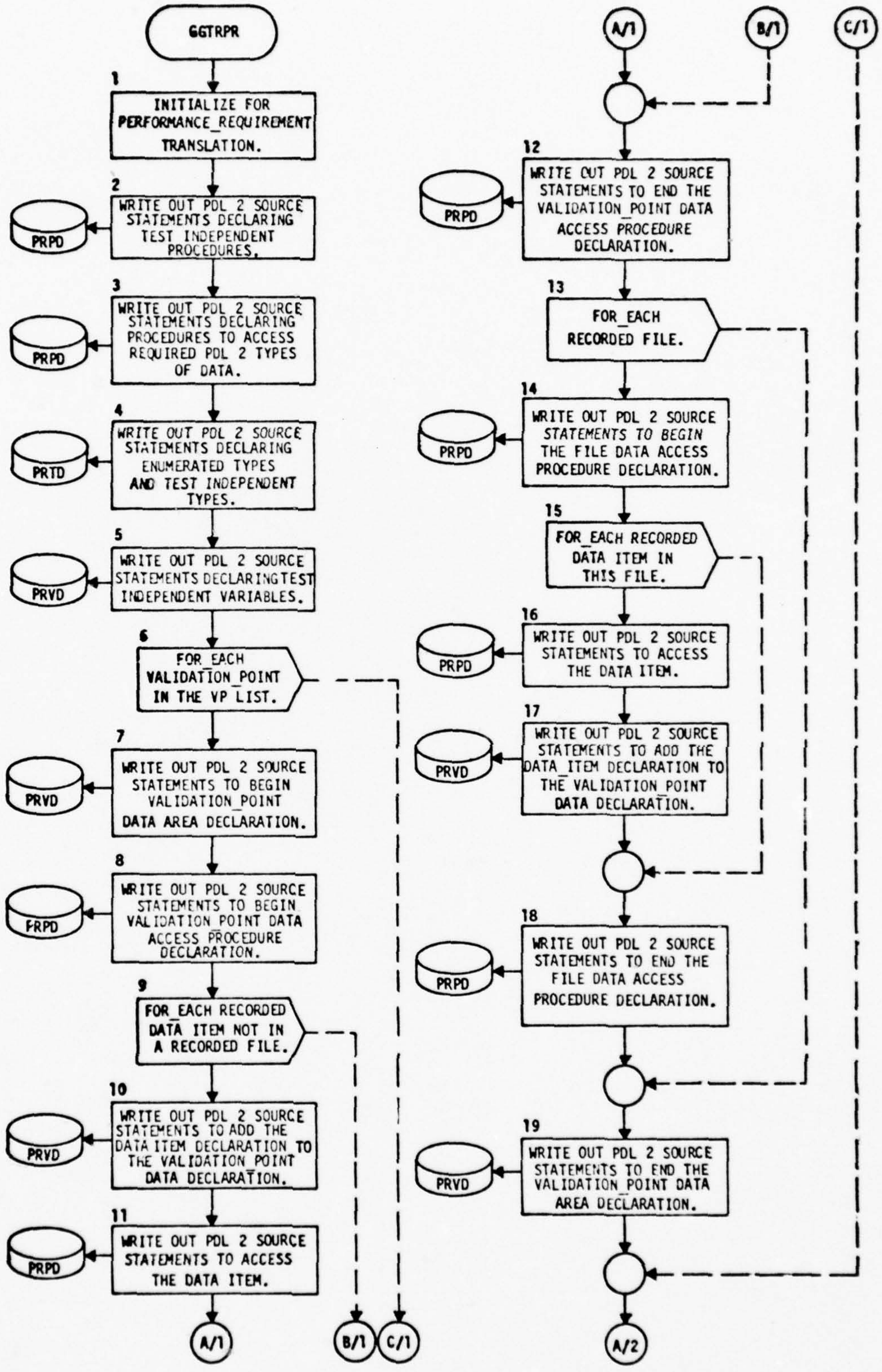



Figure 3-58.2 Performance-Requirement Translation (GGTRPR)

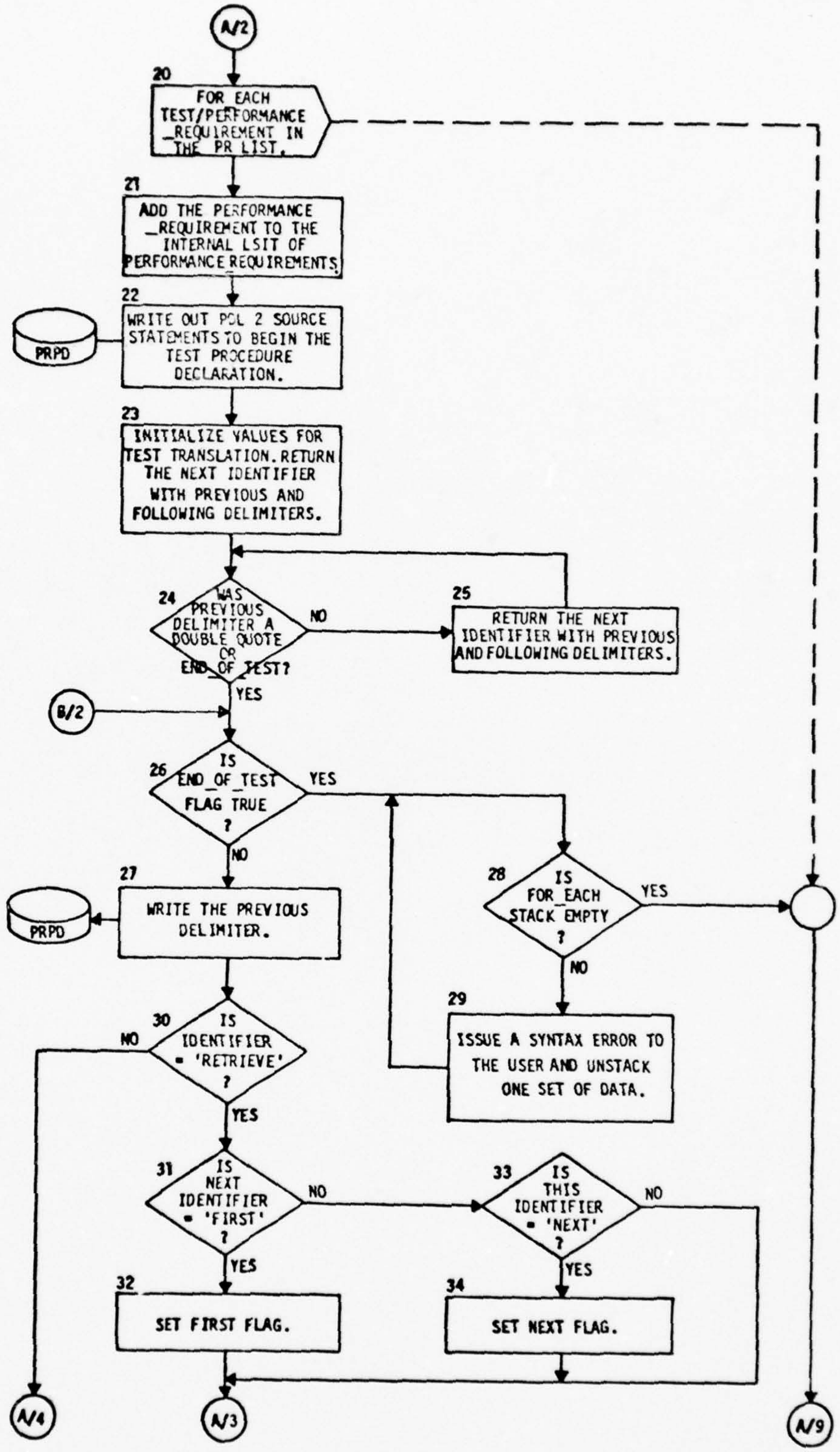



Figure 3-58.2 Performance-Requirement Translation (GGTRPR) (Continued)

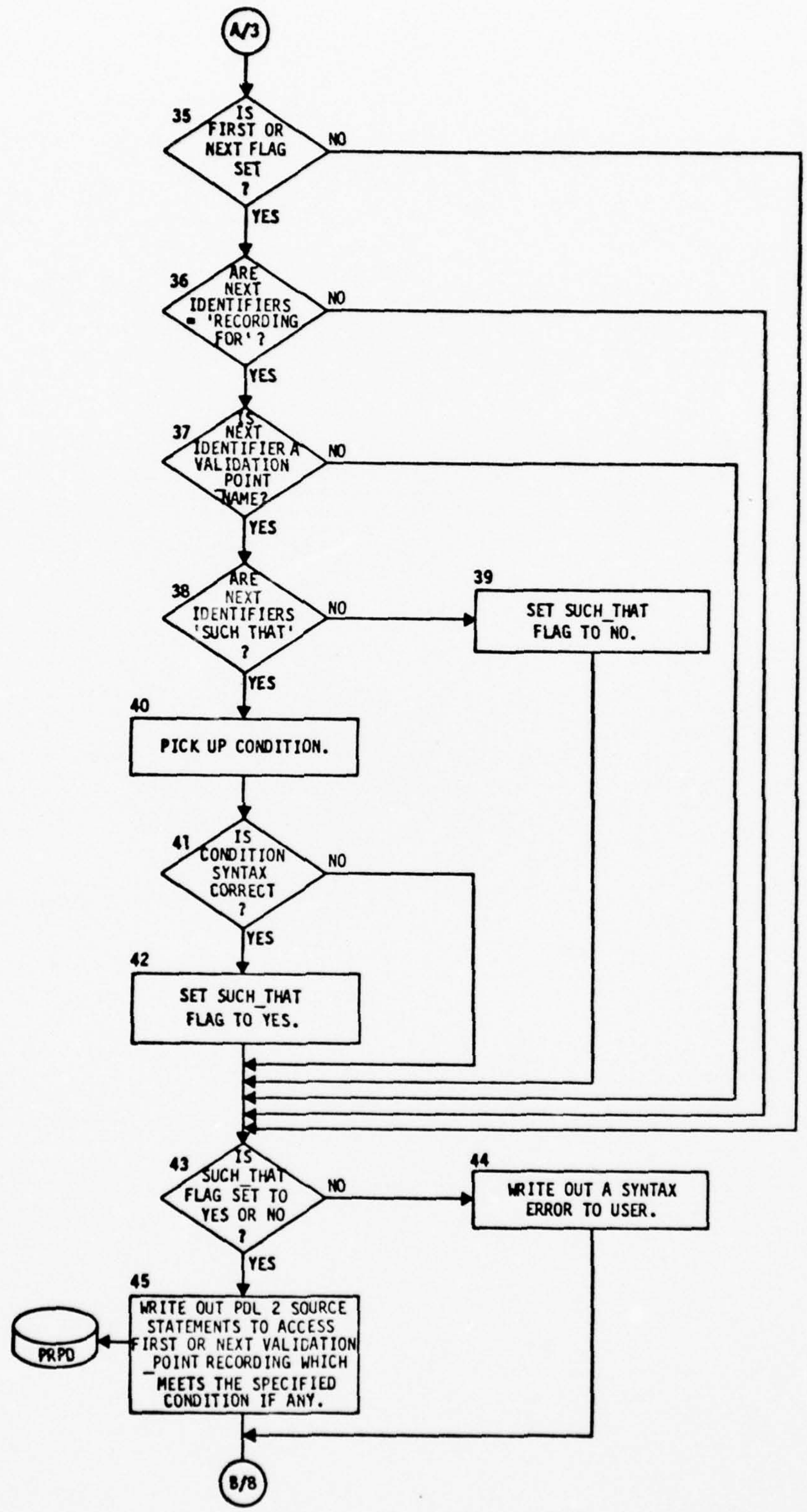



Figure 3-58.2 Performance-Requirement Translation (GGTRPR) (Continued)

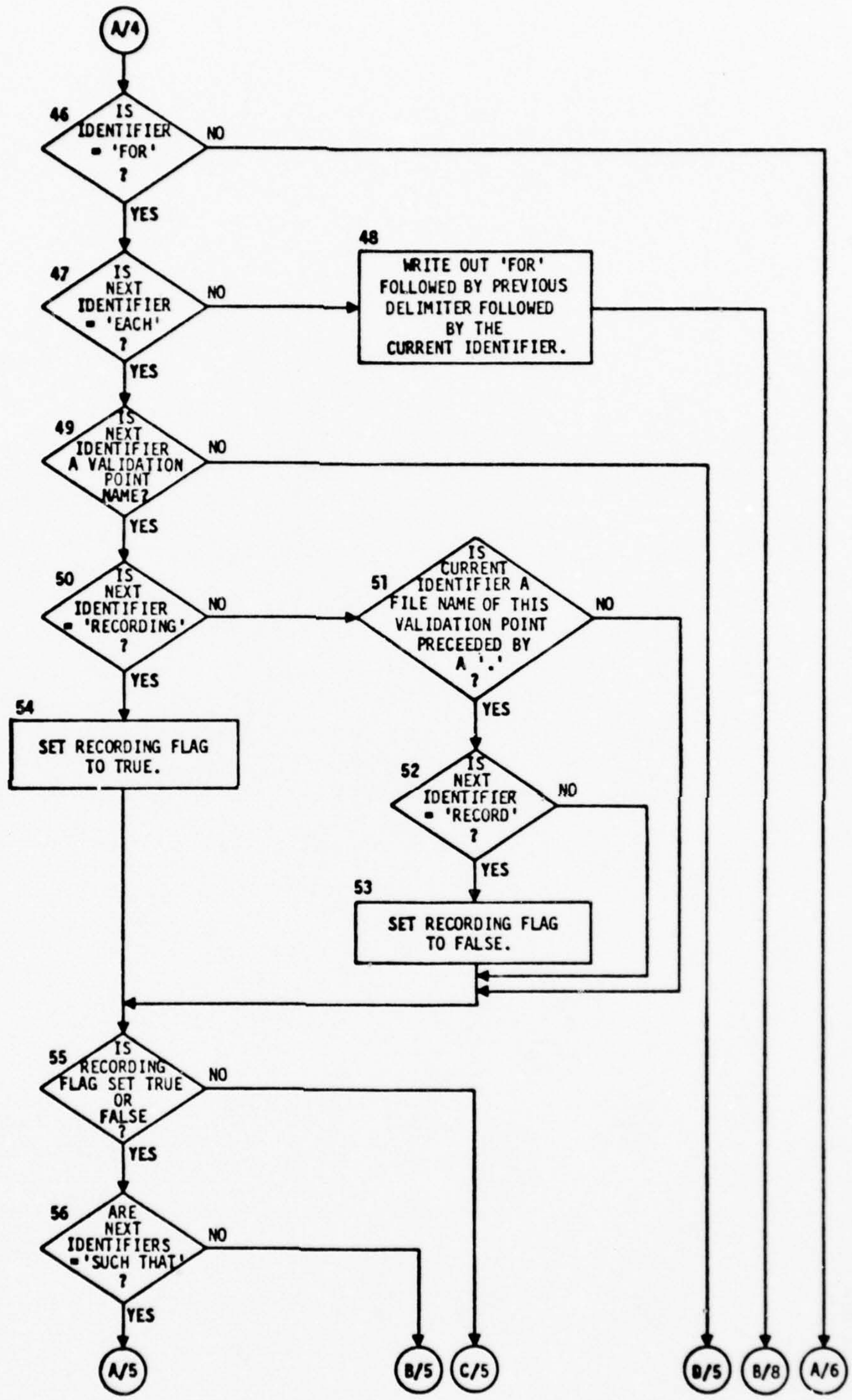



Figure 3-58.2 Performance-Requirement Translation (GGTRPR) (Continued)

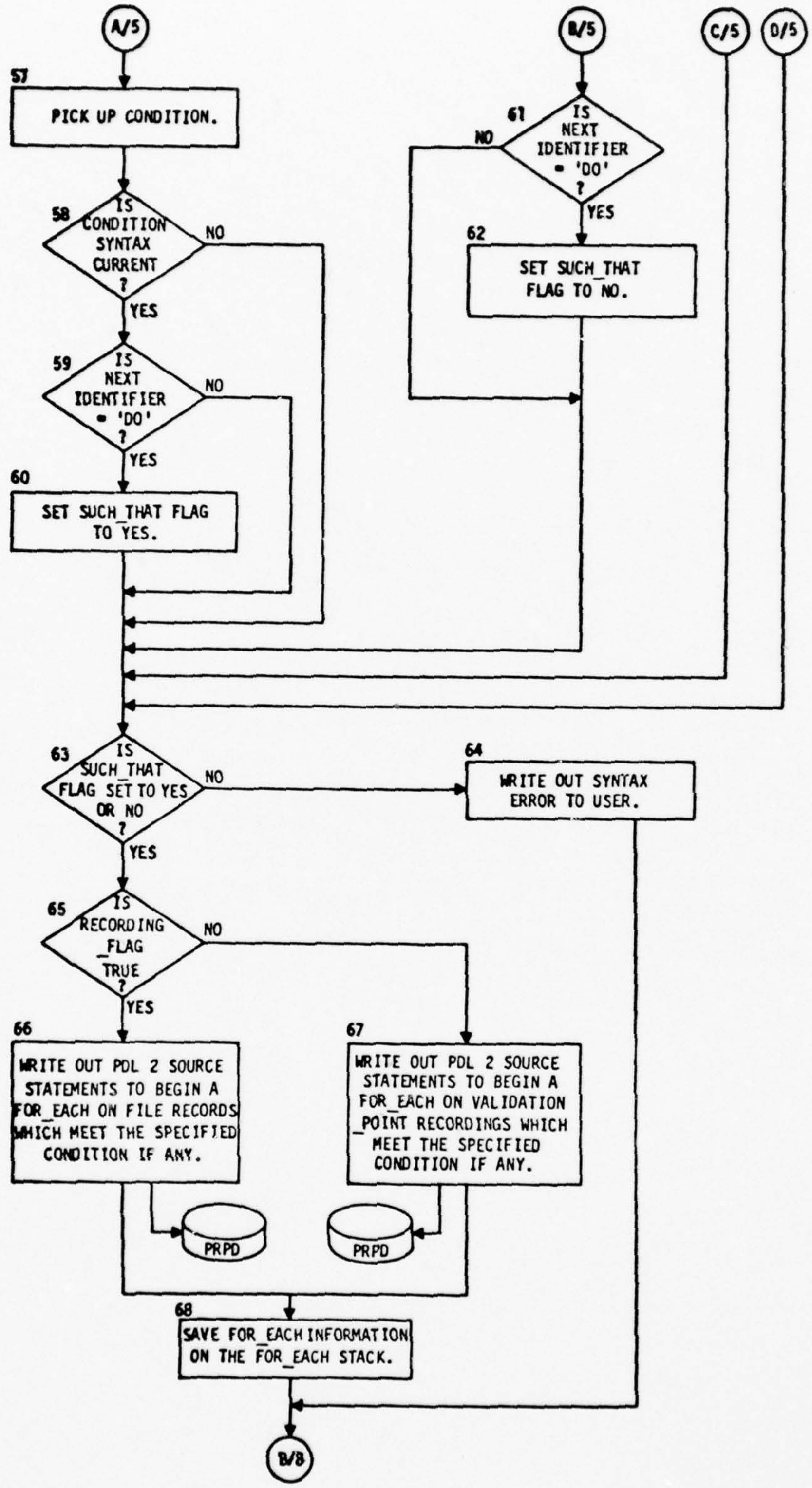



Figure 3-58.2 Performance-Requirement Translation (GGTRPR) (Continued)

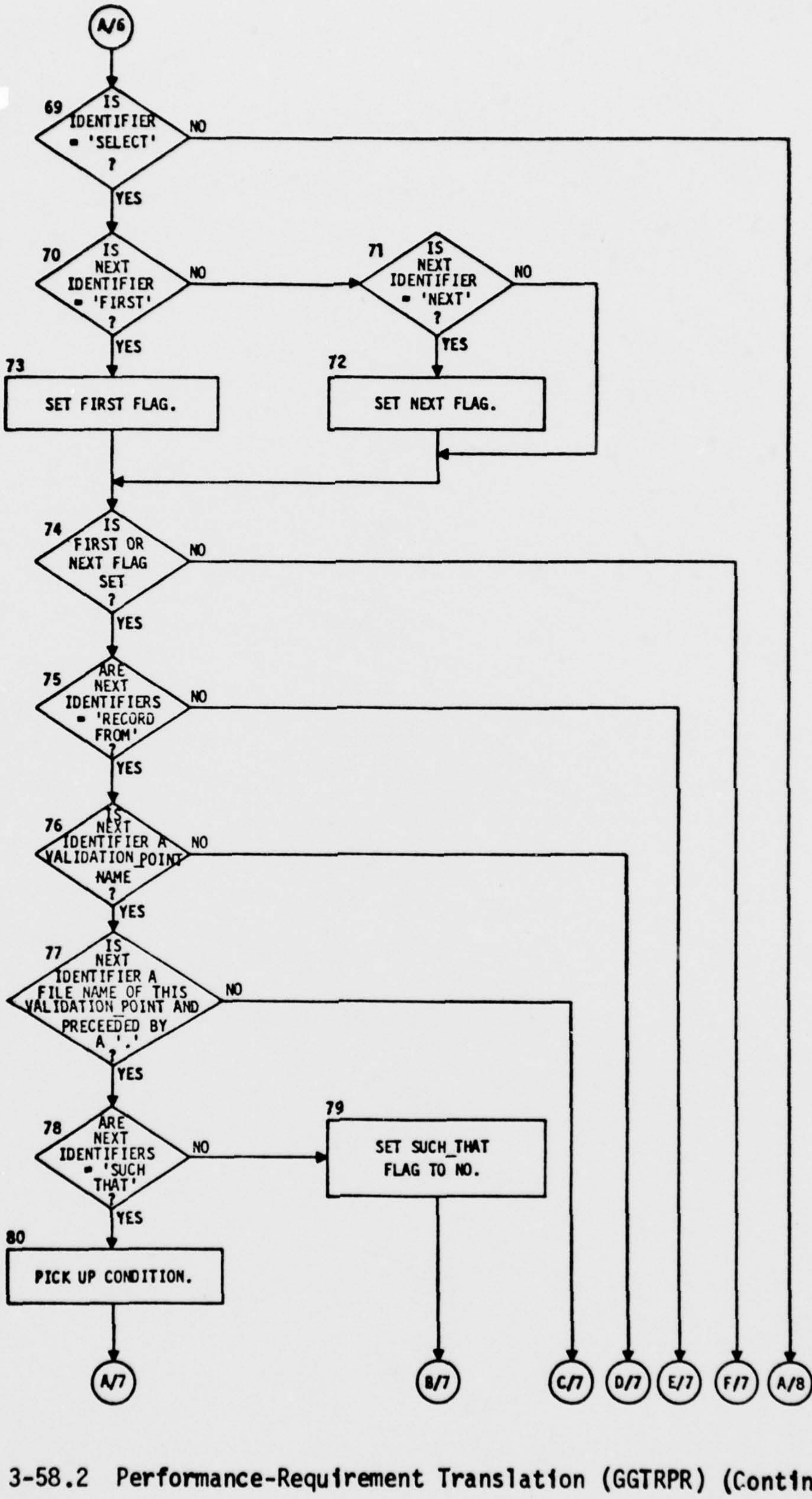



Figure 3-58.2 Performance-Requirement Translation (GGTRPR) (Continued)

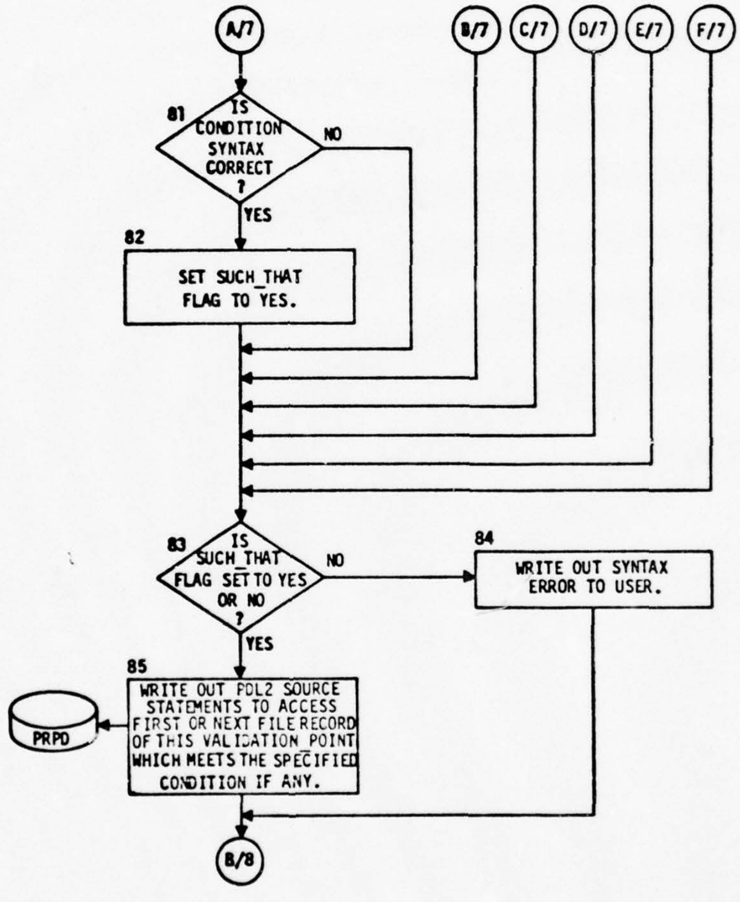



Figure 3-58.2 Performance-Requirement Translation (GGTRPR) (Continued)

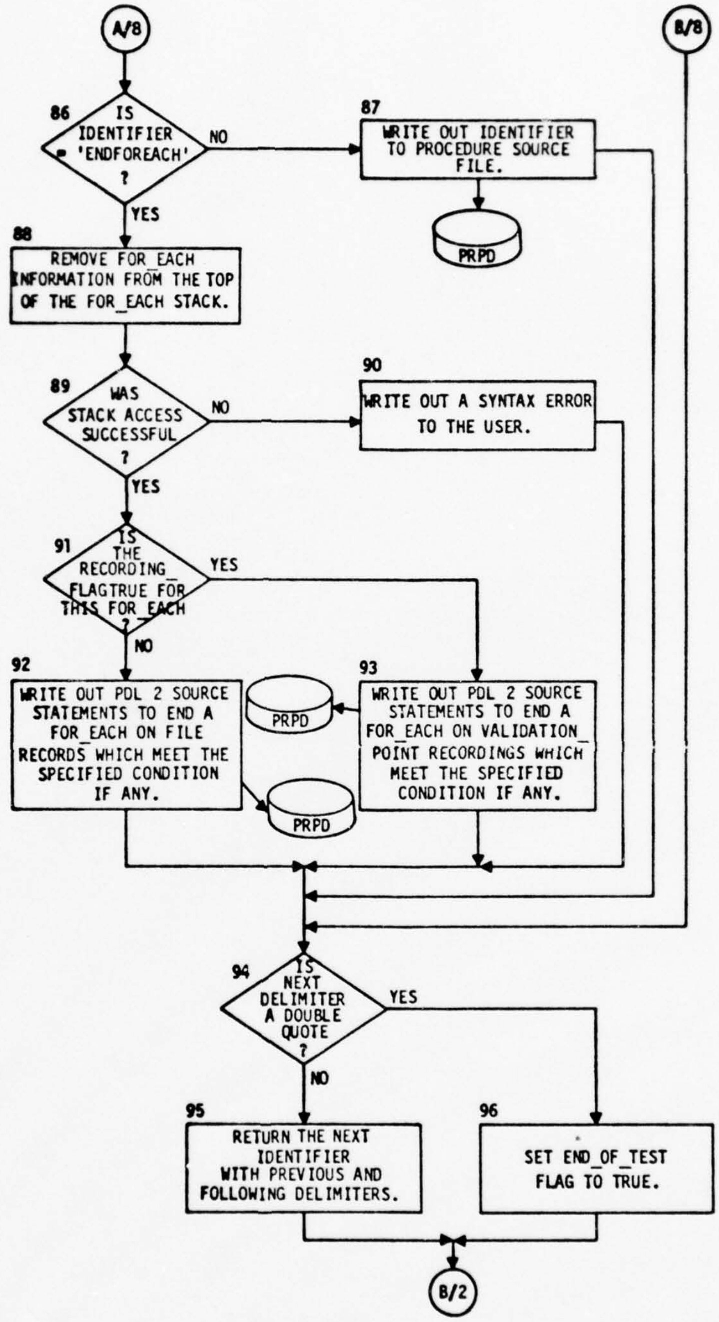



Figure 3-58.2 Performance-Reuqirement Translation (GGTRPR) (Continued)

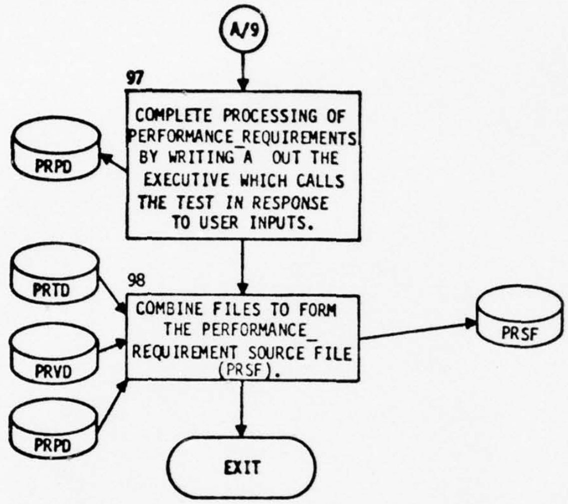



Figure 3-58.2 Performance-Requirement Translation (GGTRPR) (Continued)

## 3.6 SIMULATOR EXECUTION (SIMXQT)

### Description

The Simulator Execution function processes the simulation run-time control parameters input by the user and builds a parameter file to be read by the Simulator Program generated by Simulator Generation. The currently defined control parameters are simulation start and end times, and an identifying name for the simulator execution.

### Input

| | | |
|---|---|---|
| USER RCL | - | Simulator Execution control statements. |

### Output

| | | |
|---|---|---|
| SIMULATOR USER INPUT FILE (EESUIF) | - | Used to communicate the user inputs from SIMXQT to the Simulator Program. Contains desired simulation start and stop times, and a run identification. |

### Processing

Processing for the Simulator Execution function is shown in Figure 3-59. Additional comments which reference the processing box numbers in the flow diagram are:

| | | |
|---|---|---|
| [1] | - | Initialize parser keywords and set default run identification. |
| [3] | - | Parsing is done by a keyword driven parser. |
| [4-6] | - | Invalid statements are reported back to the user and rejected. Valid statement information is stored in internal form. |
| [7] | - | Both START and END times must be specified. |
| [9] | - | START time must be less than END time. |
| [11] | - | Valid inputs are written to the Simulator User Input File (EESUIF). |

Procedure References

The software procedures which accomplish the processing shown in the blocks of Figure 3-59 are as follows:

| | | |
|---|---|---|
| [1] | - | RRINITIAL |
| [2-6] | - | RRPARSER |
| [7-11] | - | RRWRITER |

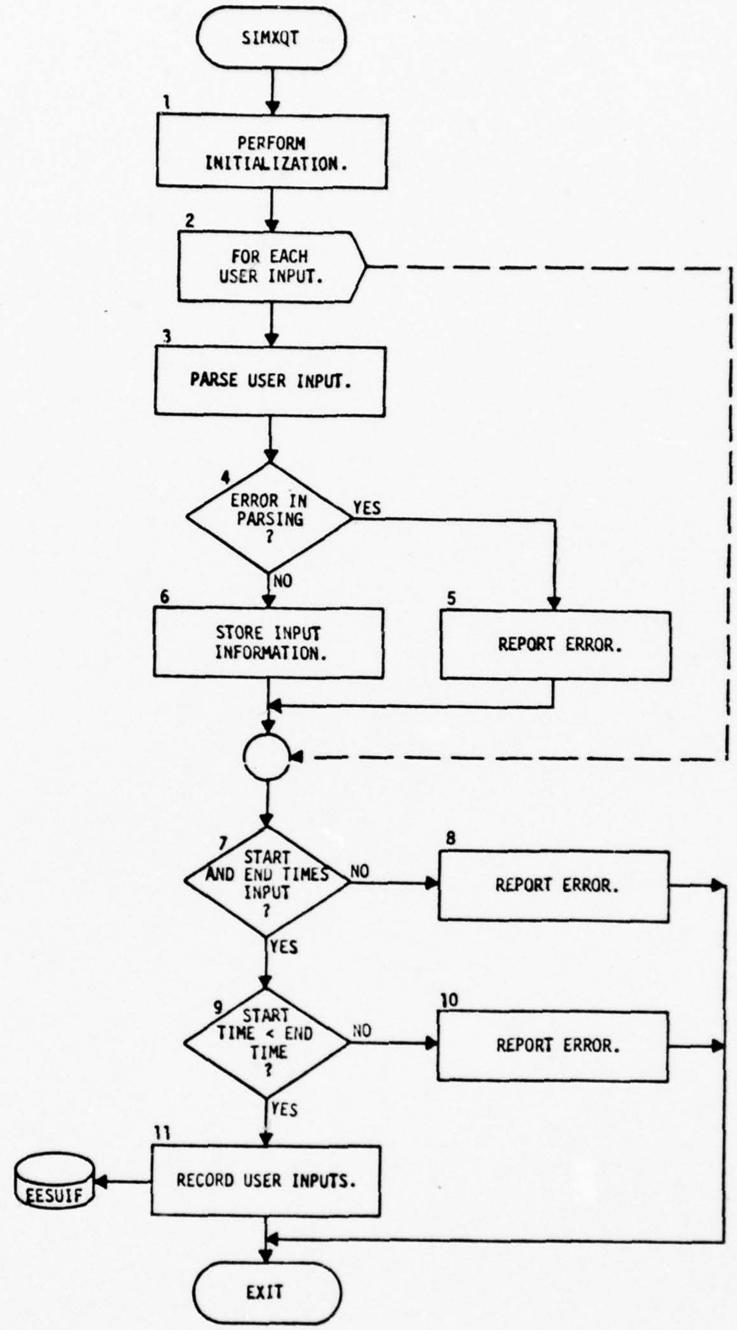



Figure 3-59 Simulator Execution (SIMXQT)

## 3.7 SIMULATION DATA ANALYSIS (SIMDA)

### Description

The Simulation Data Analysis function processes the simulation post-processor run-time control parameters input by the user and constructs a parameter file to be read by the Simulator Post-Processor Program generated by Simulator Generation. The parameter file contains the names of the PERFORMANCE_REQUIREMENTs which are to be tested by the Simulator Post-Processor Program, the time and data at which Simulator Generation was executed, and the simulation identification that the user supplied as an input to Simulator Generation.

### Input

| | | |
|---|---|---|
| USER RCL | - | Simulator Post-Processor control statements. |
| EVENT DEFINITION FILE (EEDF) | - | Information describing the Simulator and Post Processor constructed by SIMGEN. Of importance to SIMDA are the records containing the PERFORMANCE_REQUIREMENT names testable by the Post Processor and the time, date, and identification for the SIMGEN execution. |

### Output

| | | |
|---|---|---|
| VERIFICATION AND VALIDATION INFORMATION FILE (VVIF) | - | Used to communicate the user inputs and SIMGEN time, data, and identification from SIMDA to the Simulator Post-Processor Program. |

### Processing

Processing for the Simulation Data Analysis function is shown in Figure 3-60. Additional comments which reference box numbers in the flow diagram are:

| | | |
|---|---|---|
| [1] | - | The EEDF file output by SIMGEN is read to obtain the time and date SIMGEN was executed, the simulation identification, and the names of PERFORMANCE_REQUIREMENTs which can be tested by the Simulation Post-Processor Program. |
| [2-3] | - | If the EEDF file contains no PERFORMANCE_REQUIREMENT name, SIMDA reports that fact and exits. |

[5-9] - Invalid statements are reported back to the user and rejected. Valid statements are stored in internal form.

[10] - The VVIF file will contain the time, date, and ID read from the EEDF plus the names of PERFORMANCE_REQUIREMENTs to be tested.

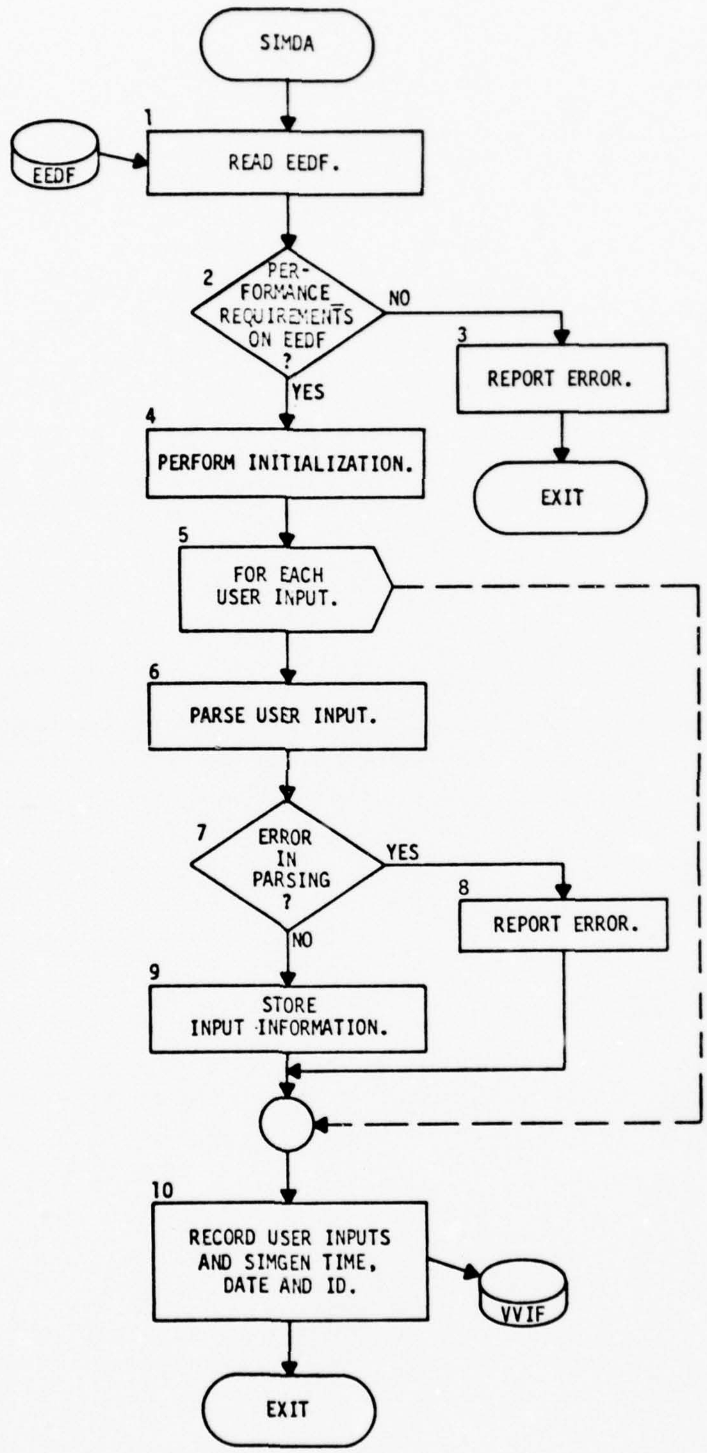



Figure 3-60 Simulation Data Analysis (SIMDA)

## 4.0 COMPUTER PROGRAM DESCRIPTION - REVS GENERATED SIMULATOR PROGRAM

Description

The Simulator Program (EEPROGRAM) is a PDL 2 main program constructed by the Simulator Generation function. Once the Simulator Program is constructed and the necessary user inputs are supplied through Simulator Execution, the program is completely executable outside the control of REVS. The Simulator Program is a discrete event type simulator. It is composed of the following major processing elements as shown in Figure 4-1.

- R_NET Procedures
- System Environment and Threat Simulator (SETS)
- Simulator Executive
- Simulator Event Management
- Simulator Data Management

Additional processors, provided externally to REVS, will generate attack descriptions compatible with the SETS model program.

The Simulator Program is designed to interface the R_NET procedures with a simulation driver such as SETS. The overall simulator control and the engagement clock resides in the Simulator Executive. SETS interfaces with the R_NET procedures through Simulator Data Management and Simulator Event Management utilities.

An R_NET is the only element in RSL which can be scheduled to execute. An R_NET is scheduled to execute whenever flow passes through an EVENT which ENABLEs the R_NET or when a MESSAGE is PASSED THROUGH an INPUT_INTERFACE which CONNECTs to the Data Processing Subsystem. SETS is scheduled to execute whenever a MESSAGE is PASSED THROUGH an OUTPUT_INTERFACE which connects to a SUBSYSTEM modeled by SETS. SETS may also enable itself through a special exogenous event. Simulator Event Management provides the utilities to schedule the execution of both R_NETs and SETS. Simulator Data Management controls and provides access to the MESSAGEs which are PASSED THROUGH the interfaces, as well as managing all other RSL DATA constructs. The Simulator Executive controls the execution of the simulator by causing the control flow to pass to SETS and the R_NETs at the scheduled times.

Input

| | | |
|---|---|---|
| SIMULATOR USER INPUT FILE (EESUIF) | - | User input controls processed by SIMXQT (See Section 3.6). |
| EVENT DESCRIPTION FILE (EEDF) | - | Definitions of the events and interfaces, and the SIMGEN time, date, and identification. |
| ATTACK SCENARIO FILE | - | Attack Scenario Definition (input to SETS). |

Output

| | | |
|---|---|---|
| VALIDATION DATA FILE (EEVALDAT) | - | Validation data recorded during a BETA simulation. |
| VALIDATION POINT RECORDING FILE (EE9VPRFL) | - | Validation data recorded during a GAMMA simulation. |

Processing

The control flow through the Simulator Program is shown in Figure 4-2. As shown in the flow diagram, control is transferred first to Simulator Initialization (EEINITIAL) and then to the Simulator Executive (EEXEC) which controls the sequencing and execution of all processing. These two functions as well as the Simulator Data Management and Event Management utilities are described in the following sections.

Procedure Reference

The processing depicted in the blocks of Figure 4-2 is performed by the following software procedures in the Simulator Program.

| | | |
|---|---|---|
| [1] | - | EEINITIAL |
| [2] | - | EEXEC |

## 4.1 SIMULATOR INITIALIZATION (EEINITIAL)

Description

The purpose of Simulator Initialization is to perform all initialization required for the execution of the simulator program.

Input

| | | |
|---|---|---|
| SIMULATOR USER INPUT FILE (EESUIF) | - | Contains simulation start and end times, and an identifying name for the run. |
| EVENT DESCRIPTION FILE (EEDF) | - | Event and interface description records constructed by SIMGEN and used to initialize Event Management control tables. Also the time and data SIMGEN was executed, and the identifying name for the simulator. |

Processing

The processing flow for Simulator Initialization is shown in Figure 4-3. The following comments further describe the processing of the indicated boxes.

| | | |
|---|---|---|
| [1] | - | Information read from the EEDF is used to initialize the event list (EEVLIST). The date and time SIMGEN generated the simulator program and the simulator identification are also read. |
| [2] | - | The Data Management Data-Set/Class Description Array is initialized and all global RSL data items are assigned their initial values. |
| [3] | - | The user input controls (simulation start and end times and run identification) are read from the EESUIF. |
| [4-5] | - | Information read from the EEDF is compared with simulator program values. If the information does not match, the EEDF and simulator program were not constructed by the same SIMGEN execution. |
| [6-7] | - | Both start time and end time must be input. |
| [8] | - | If the EEDF does not match the simulator program or if both start and end times were not input, the simulator cannot be executed. |

[9] - The simulator start-up event is posted to occur at a large negative time (-1.0E+4). The SETS-supplied procedure SSSTARTUP will execute at that time, allowing for any required SETS initialization.

[10] - The simulator shut-down event is posted to occur at simulation end time. The standard procedure EESTOP will execute at that time, resulting in program termination.

Procedure Reference

The software modules in which the processing shown in the blocks of Figure 4-3 is implemented are as follows:

[1-4] - EEINITIAL

[2] - EE8SETUP

[9-10] - EECAUSE

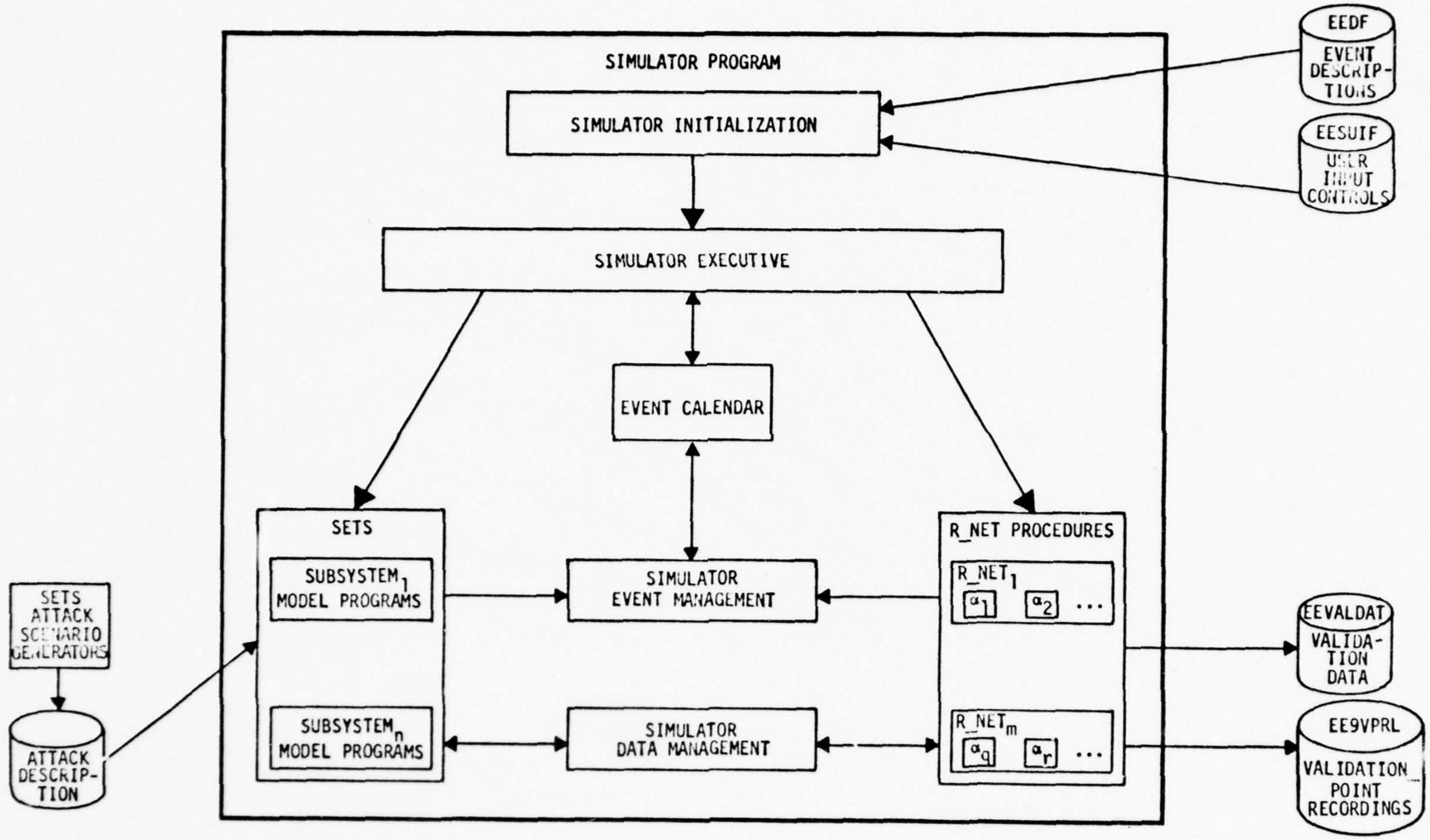



Figure 4-1 Simulator Program Overview

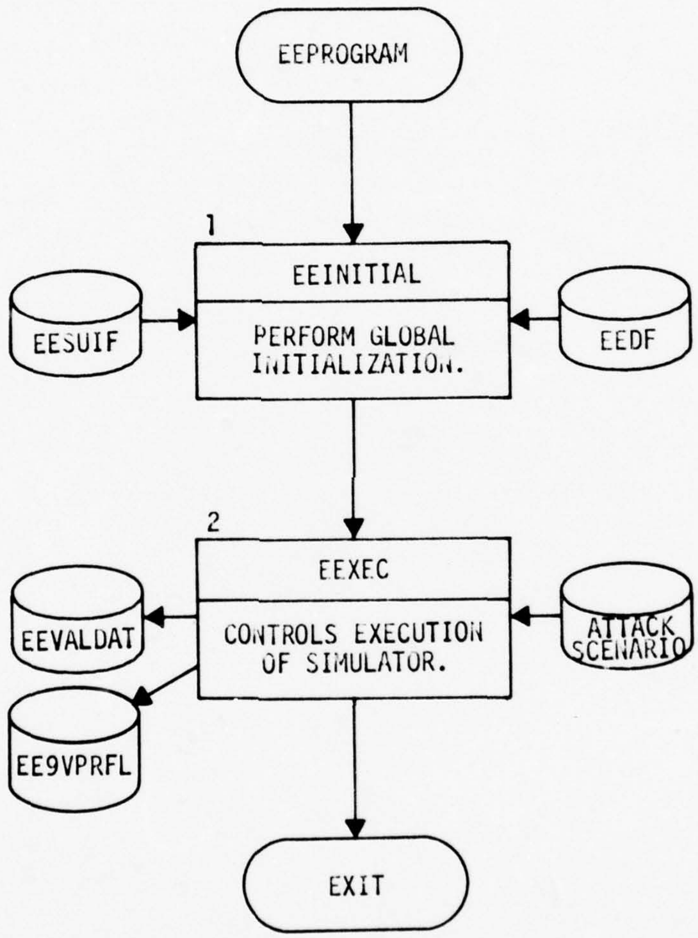



Figure 4-2 Simulator Program (EEPROGRAM)

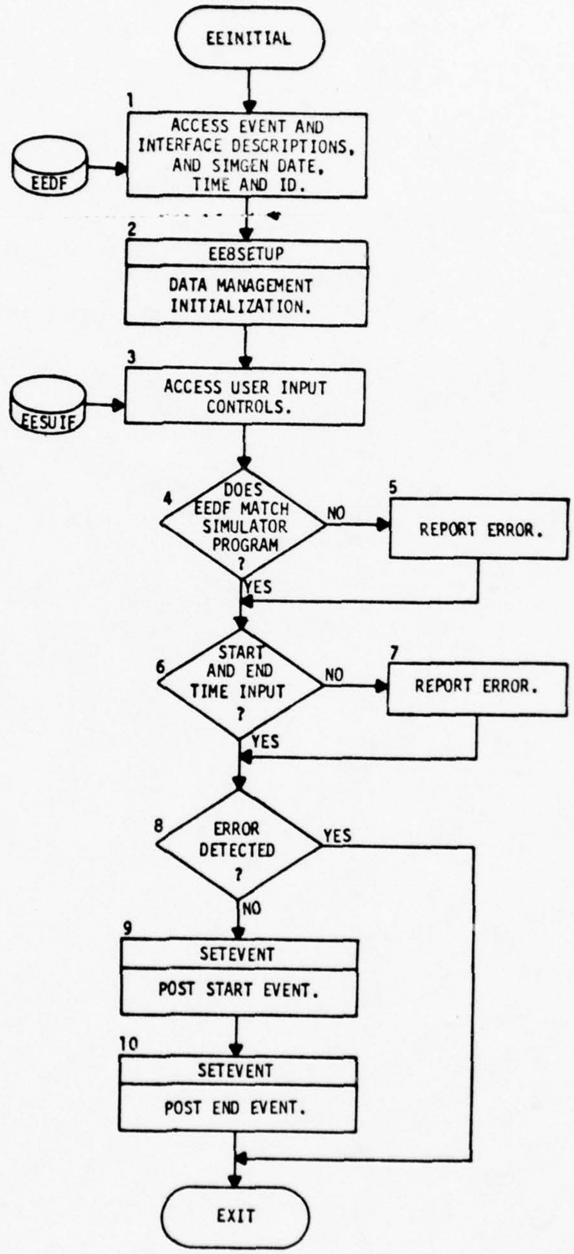



Figure 4-3 Simulator Initialization (EEINITIAL)

## 4.2 SIMULATOR EXECUTIVE (EEXEC)

### Description

The Simulator Executive controls the sequence of execution of R_NET and SETS models and maintains the engagement clock. The execution sequence is determined by the order of events on the event calendar and by the enablement conditions for the models. The event calendar is a time-ordered linked list constructed and accessed using the Simulator Event Management utilities (see Section 4.3). The Simulator Executive removes one event at a time from the linked list, updates the engagement time to the event time, and invokes any and all R_NETs or SETS models which are fully enabled. cycling through the event list continues until the termination flag is set. This flag is set true when the standard "pseudo" R_NET EESTOP executes at engagement end time or in case the event list becomes empty.

### Input

| | | |
|---|---|---|
| EVENT LIST (EEVLIST) | - | A matrix describing each event in the simulation. The content is the same as an event definition record on the EEDF. |
| EVENT CALENDAR | - | A time-ordered linked list of events which are scheduled to occur. |
| EVENT/ENABLEMENT DEPENDENCY LIST (EEDEPLST) | - | A matrix describing the dependency of R_NET enablements on events. |

### Output

| | | |
|---|---|---|
| EVENT/ENABLEMENT DEPENDENCY LIST (EEDEPLST) | - | A matrix describing the dependency of R_NET enablements on events. |
| EVENT CALENDAR | - | A time-ordered linked list of events which are scheduled to occur. |

### Local Data

| | | |
|---|---|---|
| TERMINATION FLAG | - | A flag which is set true by EESTOP at engagement end time or is set true when the event calendar is empty. |

Processing

The processing performed by the Simulator Executive is shown in Figure 4-4. Additional comments referencing these boxes are given below.

| | | |
|---|---|---|
| [1] | - | Termination flag will be set by procedure EESTOP at simulation end time. |
| [2, 3] | - | An empty event calendar will cause the termination flag to be set. |
| [4] | - | Engagement time is set equal to the event time. |
| [5] | - | The Event List (EEVLIST) entry for the current event is used to update the Event/Enablement Dependency List (EEDEPLST). |
| [6-9] | - | This code is constructed by the Event Translation phase of Simulator Generation. |

Procedure Reference

The software procedures which accomplish the processing shown in the blocks of Figure 4-4 are as follows:

| | | |
|---|---|---|
| [1] | - | EEXEC |
| [2-4] | - | EERTOPE |
| [5] | - | EESETDEP |
| [6-9] | - | EESCHED |

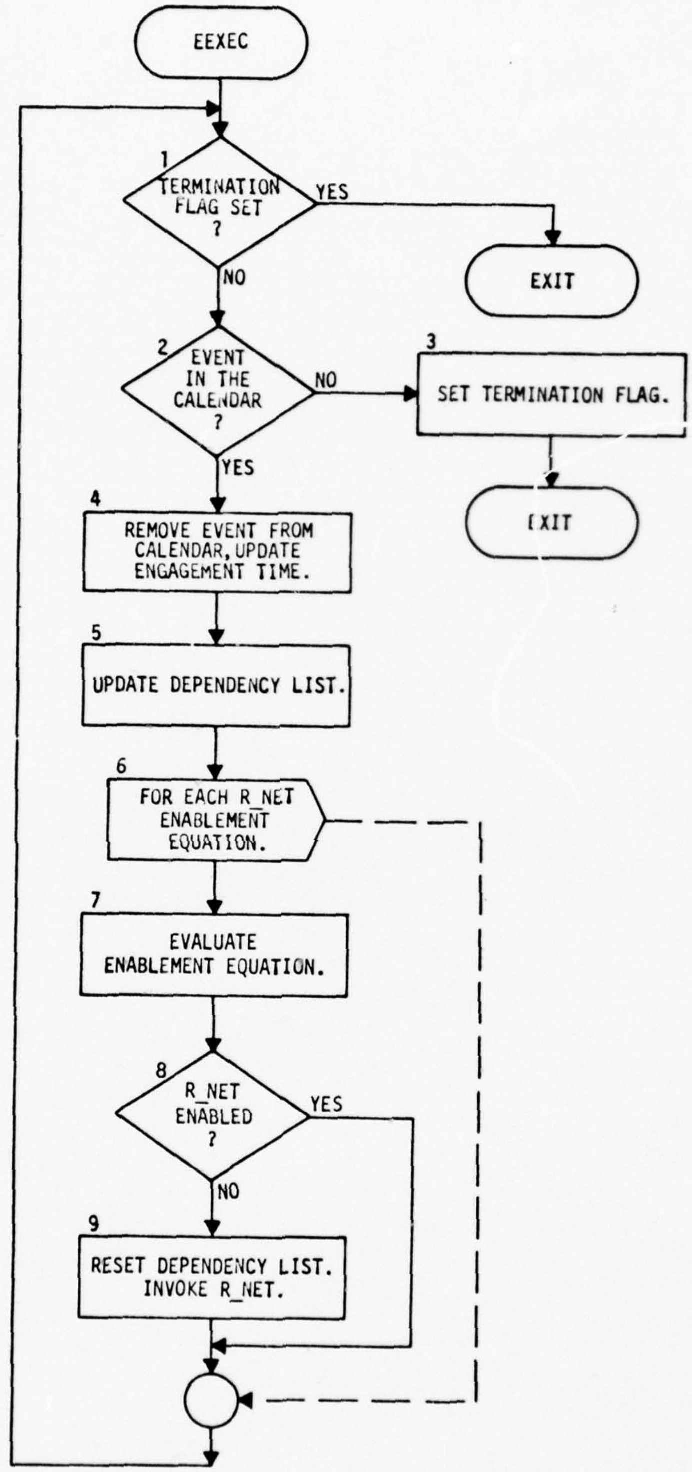



Figure 4-4 Simulator Executive (EEXEC)

## 4.3 SIMULATOR EVENT MANAGEMENT

The Simulator Event Manager provides the utilities necessary to correctly maintain the event calendar, which is a time-ordered linked list of events scheduled to occur. Two conceptual operations are permitted upon the event calendar, insertion of an event (in its proper time-determined position) and removal of the top or next event to occur in time order. The insertion of an event is accomplished by the SETEVENT utility which has three callable interfaces; EESETEV, EECAUSE, and EECAUSED. The removal of the next event is accomplished by the REMOVE utility which is invoked by calling the procedure EERTOPE. These utilities are described in the following sections.

### 4.3.1 SETEVENT

Description

As stated above, the SETEVENT utility can be invoked through three procedures, EESETEV, EECAUSE, and EECAUSED. EESETEV is the standard procedure used by the requirements dependent code constructed by Simulator Generation. The EESETEV procedure is thus called whenever control flow passes through an EVENT or OUTPUT_INTERFACE. The EECAUSE and EECAUSED procedures are provided mainly for use by SETS models. They provide a simpler calling sequence while maintaining the capability for setting events at the current engagement time (EECAUSE) or at some delayed time (EECAUSED).

Input

| | | |
|---|---|---|
| EVENT CALENDAR | - | Time-ordered linked list of scheduled events. Each record in the list contains an event number, an event time, and a link to the next record. |
| EVENT NAME | - | RSL name for the EVENT or INTERFACE. |
| EVENT NUMBER | - | Number assigned by SIMGEN to the event. (Not supplied for EECAUSE or EECAUSED.) |
| EVENT TIME | - | Time at which the event is to occur. |

Local Data

FIRST EVENT - Pointer to top of the event calendar.

LAST EVENT - Pointer to last event on the calendar.

Output

EVENT CALENDAR - Time-ordered linked list of scheduled events.

Processing

The processing performed by the SETEVENT utility is shown in Figure 4-5. Additional comments concerning selected boxes on the flow diagram are provided below.

[1,2] - Unknown events or events which would cause engagement time to back up cannot be scheduled.

[8-10] - The event is placed after any other event on the calendar with the same time. Thus, events at a particular time are treated in a FIFO (First-In-First-Out) manner.

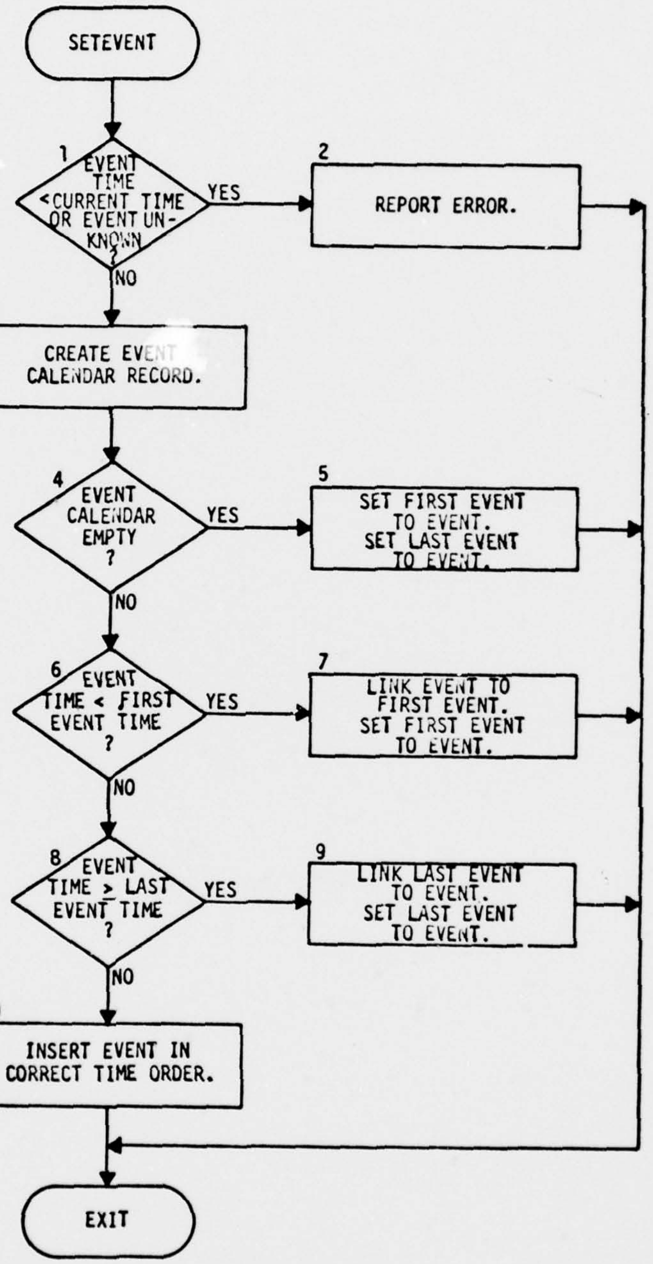



Figure 4-5 SETEVENT

### 4.3.2 Remove Event (EERTOPE)

Description

The Remove Event utility is used by the Event Manager to retrieve the next scheduled event from the event calendar and update the engagement clock to the time of this event. This utility is only used by the Simulator Executive and is accomplished by the procedure EERTOPE.

Input/Output

| | | |
|---|---|---|
| ENGAGEMENT TIME (EECLOCK) | - | Current engagement time. |
| EVENT CALENDAR | - | Time-ordered linked list of scheduled events. |
| FIRST EVENT | - | Pointer to first event on calendar. |
| CURRENT EVENT | - | Pointer to current event. |

Local Data

| | | |
|---|---|---|
| TERMINATION FLAG | - | Flag which is set true if the event calendar is empty. |

Processing

The processing performed by the REMOVE utility is shown in Figure 4-6.

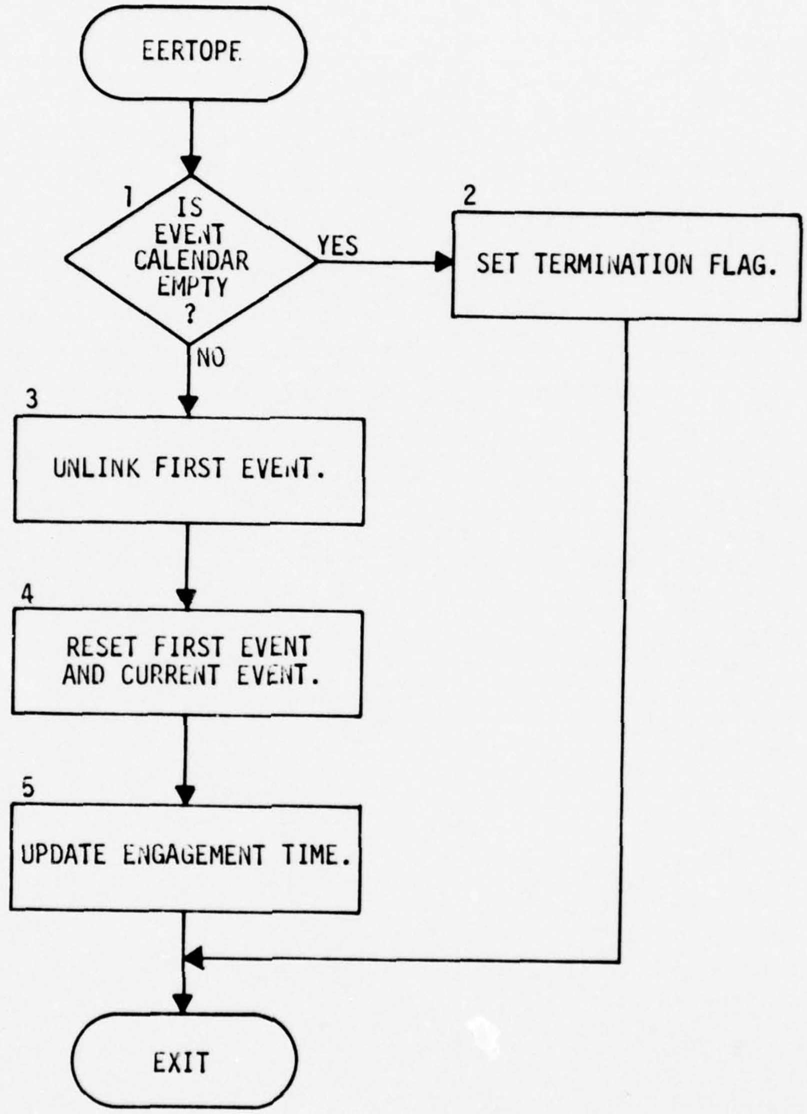



Figure 4-6 Remove Event (EERTOPE)

## 4.4 SIMULATOR DATA MANAGEMENT

The Simulator Data Manager provides the service requests specified in the ALPHA models and R_NETS translated from RSL. Specifically, it supports INTERFACE, FILE, and ENTITY accesses using list processing techniques in either First-In-First-Out (FIFO) order or Rank High order (FILEs only).

The service requests supported are:

- creation of an instance
- destruction of an instance
- transformation of an instance of one type to an instance of another type (ENTITY_TYPE only)
- formation of a MESSAGE instance for INTERFACE transmission
- access to first instance
- access to next instance
- access all instances successively.

The Simulator Data Manager is actually a library of procedures which implement the required service requests by manipulating linked list structures in dynamic memory and the corresponding simple variables in static memory. The name of each data-set (FILE, INTERFACE, or ENTITY_TYPE) and class (ENTITY_CLASS) is declared by the Simulator Generation function as an index to a Data-Set or Class Description Block. The Data-Set Description Block forms the header for the linked list of instances owned by the data-set. Data Management service calls are translated into procedure invocations which pass the data-set name as a parameter and which access the linked lists via the indexed Data-Set Description Block.

A user can only access a DATA value after it has been transferred from the dynamic record (linked in a list) to a static variable by a Data Management service call. The user can modify the DATA values in the static variables as long as the instance is selected. The selection of another instance or the creation of a new instance will cause the currently selected instance to be updated. That is, the static variables (containing possibly altered DATA values) will be transferred back to the dynamic record in the linked list. For ENTITY_TYPEs an instance may also own FILEs

as well as DATA values. An owned FILE can only be accessed while the owning instance is selected. This is accomplished by transferring the FILE description block to and from the dynamic instance record just as a DATA value.

When a user creates an instance of a data-set or class, the Data Manager will initialize each contained data item to the initial value assigned in the ASSM or to a default initial value determined by the data item's type. A dynamic record is not created and linked into the data-set list until the instance is updated. A status flag in the Data-Set Description Block indicates whether the currently selected instance is new or old and tells the UPDATE module what actions to take on the instance.

When a user destroys an instance of a data-set or class, the Data Manager will initialize each contained data item to the initial value assigned in the ASSM or to a default initial value determined by the data item's type. The dynamic record, if one exists, is marked for deletion at a later time.

The components of the Data Manager are shown in Figure 4-7. In general those components correspond with service requests with the exception of the UPDATE component. This component is called by other components to store the current data-item values into a dynamically allocated instance record. The user is never required to make an UPDATE service request.

The following data structures are used by the Data Manager to represent and manipulate the user's data base. The Data-Set/Class Description Array (EE9DS) contains an execution time description of all data-sets and classes. Some of its characteristics are:

- declaration generated by the Data Translation module of the Simulator Generation function.
- array elements are Data-Set and Class Description Blocks.
- data fields of Data-Set/Class Description Array are set at simulation initialization by a procedure generated by the Data Translation module of the Simulator Generation function.
- data-sets owned by the same class have description blocks which are contiguous in the Data-Set/Class Description Array.

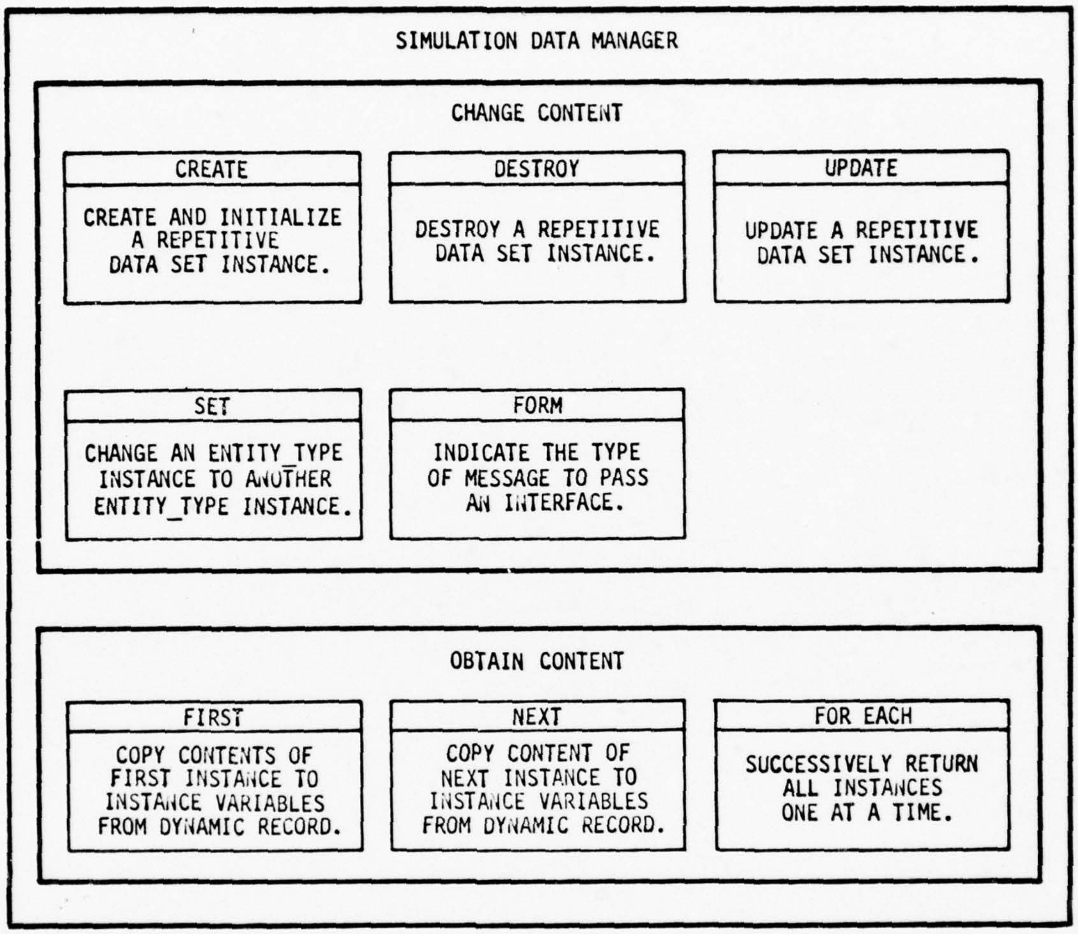



Figure 4-7 Simulation Data Manager Components

- the Class Description Block immediately follows the Data-Set Description Blocks which it owns.
- the Data-Set Description Blocks and Class Description Blocks are of the same type record (EE7DSTYP) with variants declared to distinguish the two different sets of DATA fields. The DATA field DSKIND establishes the type of variant (data-set or class).

The following identifies the data fields contained in a Data-Set Description Block in the Data-Set/Class Description Array:

| | | |
|---|---|---|
| DATA-SET/CLASS FLAG (DSKIND) | - | Set to 'Data-Set' for a Data-Set Description Block in the Data-Set/ Class Description Array. |
| INDEX TO DATA-SET (DSTYP) | - | Index into the Data-Set/Class Description Array for this data-set entry. |
| POINTER TO FIRST INSTANCE OF DATA-SET (BOI) | - | Set to 'NIL' if data-set is empty. |
| POINTER TO CURRENTLY SELECTED INSTANCE (CIN) | - | Set to 'NIL' if no instance is selected. |
| POINTER TO LAST INSTANCE OF DATA-SET (EOI) | - | Set to 'NIL' if data-set is empty. Set to same value as BOI for one instance in data-set. |
| DATA-SET STATUS (INSTT) | - | Set to 'old' if old instance selected. Set to 'new' if newly created instance. Set to 'null' if no instance is selected. |
| INDEX TO CLASS OWNING THIS DATA SET (CLASS) | - | Index into the Data-Set/Class Description Array entry for the owning class. |
| INSTANCE TYPE (INTYP) | - | Element from the list of instances which this data-set contains. Used to indicate which MESSAGE was FORMed for an INTERFACE. |
| DATA-SET LINK (NXTDS) | - | (Not used for Data-Set Description Blocks in Data-Set/Class Description Array.) |

The following data fields are contained in a Class Description Block in the Data-Set/Class Description Array:

| | | |
|---|---|---|
| DATA-SET/CLASS FLAG (DSKIND) | - | Set to 'Class' for a Class Description Block. |
| INDEX TO CLASS (DSTYP) | - | Index into the Data-Set/Class Description Array for this class entry. |
| INDEX TO FIRST DATA-SET OF THIS CLASS (BOC) | - | Index into the Data-Set/Class Description Array. |
| INDEX TO THE CURRENTLY SELECTED DATA-SET OF THIS CLASS (CDS) | - | Index into the Data-Set/Class Description Array. |
| INDEX TO THE LAST DATA-SET OF THIS CLASS (EOC) | - | Index into the Data-Set/Class Description Array. |
| NEW CLASS INSTANCE FLAG (NEW) | - | Set 'true' for a newly created instance which has had no type set for it. Set 'false' otherwise. |

The dynamic instance record holds the DATA values for the instance during those times when the instance is not selected. All dynamic instance records are of the same PDL 2 type with a variant declared for each distinct instance type. In order to accommodate the same DATA item in more than one instance (variant), each variant is declared further to be a record itself. The data fields which are common to all instance types are described below:

| | | |
|---|---|---|
| FORWARD INSTANCE LINK (FLNK) | - | Points to the next instance in a data-set. Set to 'NIL' for the last instance of a data-set. |
| REVERSE INSTANCE LINK (RLNK) | - | Points to the previous instance in a data-set. Set to 'NIL' for the first instance of a data-set. |
| DATA-SET LINK (DSLNK) | - | Points to the first Data-Set Description Block of a data-set owned by the instance. The data field NXTDS is used to link subsequent Data-Set Description Blocks together. |
| INSTANCE TYPE (INTYP) | - | Contains the type of instance that this record represents. |

DATA-SET TYPE (DSTYP) - Index into the Data-Set/Class Description Array for the data-set which owns the instance.

FOR EACH EXAMINATION COUNTER (CNTR) - Holds the number of FOR EACH executions which are currently examining the instance. A dynamic record cannot be delinked and disposed until CNTR equals zero.

DELETED FLAG (DLTD) - Indicates that the dynamic record is logically deleted from the linked list. The dynamic record cannot be removed from the list until no FOR EACH execution is examining the instance.

### 4.4.1 CREATE (EE8CREATE)

Description

The CREATE module of the Data Manager is called during simulator execution by a user service request to establish a new instance of a FILE or ENTITY_CLASS. First, any previously selected or newly created instance is updated (see the UPDATE module, Section 4.4.8), then the static variables associated with the new instance are set to their initial values. In addition, for ENTITY_CLASSes, all FILEs associated with the class have the static variables associated with the FILE instance set to their initial values. Finally the FILE or ENTITY_CLASS status flag will be set to 'new' to indicate that a new instance must be processed at UPDATE time.

Input

Input to the CREATE module of the Data Manager is an element of the list of data-sets/classes. The input element acts as an index to the desired data-set (FILE or ENTITY_TYPE) or class (ENTITY_CLASS) entry in the Data-Set/Class Description Array. See Section 4.4 for a description of the Data-Set/Class Description Array.

Processing

Processing performed in the CREATE module is shown in the flow diagram of Figure 4-8. The following comments refer to processing box numbers.

| | | |
|---|---|---|
| [1] | - | See UPDATE module (Section 4.4.8) for detailed description. |

Procedure Reference

| | | |
|---|---|---|
| [1] | - | EE8UPDATE |
| [4] | - | EE8ININ |
| [7] | - | EE8ININ |
| [8] | - | EE8INCL |

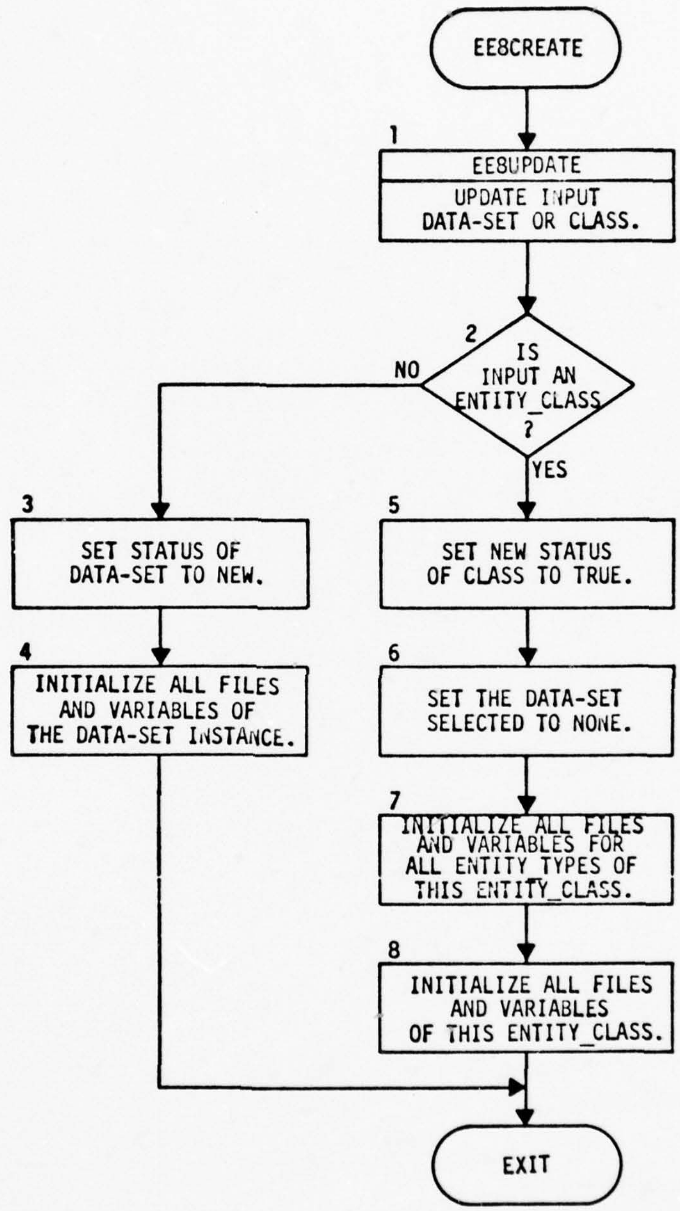



Figure 4-8 CREATE (EE8CREATE)

### 4.4.2 DESTROY (EE8DESTROY)

Description

The DESTROY module of the Data Manager is called during simulator execution by a user service request to dispose of the currently selected instance of a FILE or ENTITY_CLASS. If a DESTROY is attempted with no valid instance selected, a diagnostic message will be issued. A DESTROY marks as deleted the currently selected instance's dynamic record (if one exists) and initializes all instance static variables. For ENTITY_CLASS instances, all FILEs owned will have all instances disposed and their corresponding instance static variables initialized. The status of the data-set which has had a DESTROY operation done on it, will have a value of null.

Input

Input to the DESTROY module of the Data Manager is an element of the list of data-sets/classes. The input element acts as an index to the desired data-set (FILE) or class (ENTITY_CLASS) entry in the Data-Set/Class Description Array. See Section 4.4 for a description of the Data-Set/Class Description Array.

Processing

Processing performed in the DESTROY module is shown in the flow diagram of Figure 4-9. The following comments refer to processing box numbers.

| | | |
|---|---|---|
| [6, 15] | - | Diagnostic message is written to external file OUTPUT. |
| [8] | - | Input to this call to the DESTROY module is the data-set (ENTITY_TYPE) which is currently selected on the originally input ENTITY_CLASS. |

Procedure Reference

| | | |
|---|---|---|
| [5] | - | EE8ININ |
| [6] | - | EE8LSTDIAG |
| [8] | - | EE8DESTROY |
| [9, 13] | - | EE8ININ |
| [10, 14] | - | EE8INCL |

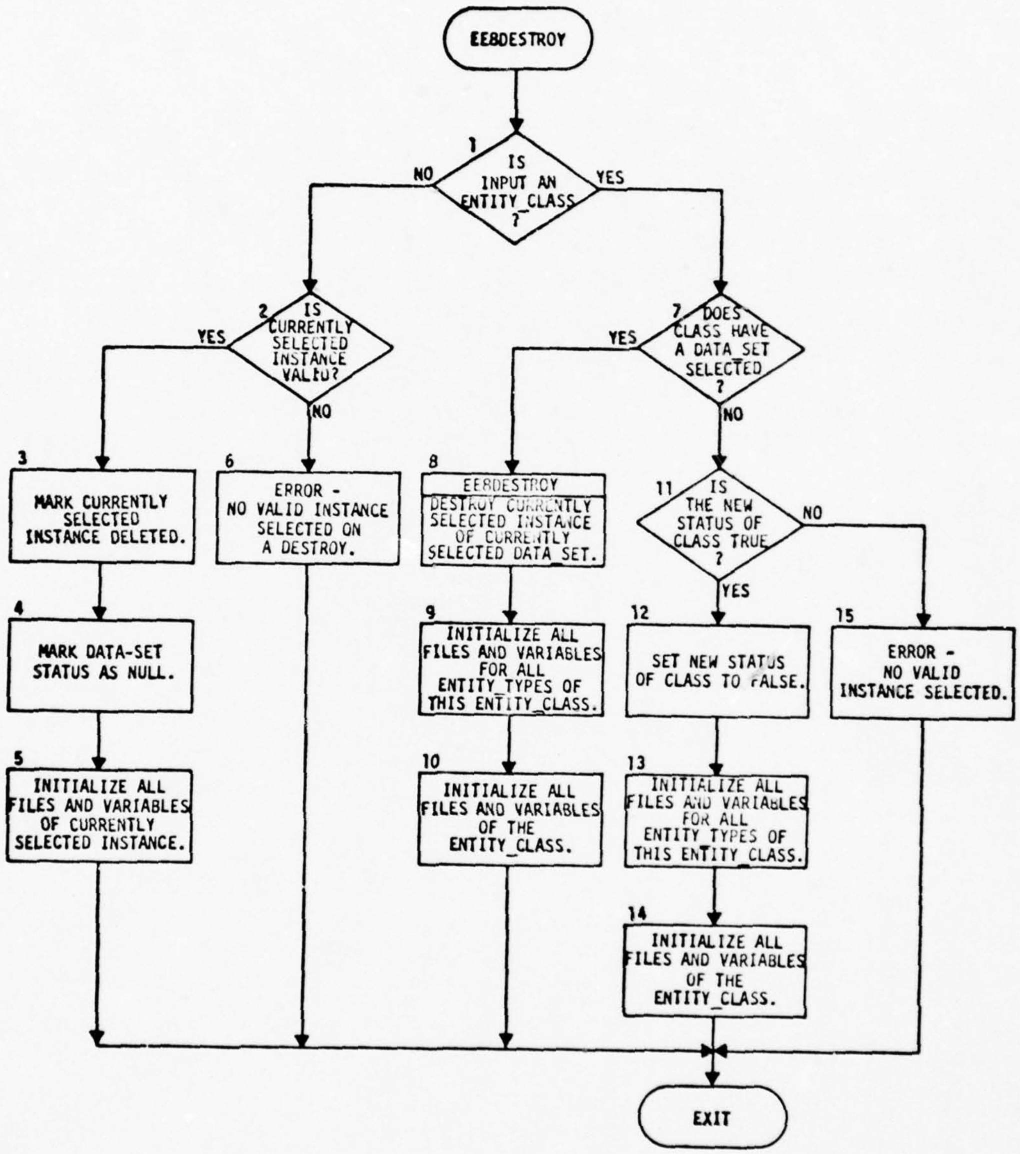



Figure 4-9 DESTROY (EE8DESTROY)

### 4.4.3 FORM (EE8FORM)

#### Description

The FORM module of the Data Manager is called during simulator execution by a user service request to establish the MESSAGE that will pass an INTERFACE when that INTERFACE is encountered. If many FORMs are executed for different MESSAGEs passing the same INTERFACE, only the last one executed before encountering the INTERFACE will be effective. An INTERFACE which is encountered with no MESSAGE FORMed on it will cause an execution time diagnostic.

#### Input

Input to the FORM module of the Data Manager is an element of the list of instances. The input element causes the associated data-set (INTERFACE) to have its instance type set to the input element and its status to be set to new.

#### Processing

Processing performed in the FORM module is shown in the flow diagram of Figure 4-10. The following comments refer to processing box numbers.

| | | |
|---|---|---|
| [1, 2] | - | Processing described in these boxes uses the Data-Set/Class Description Array (see Section 4.4). |

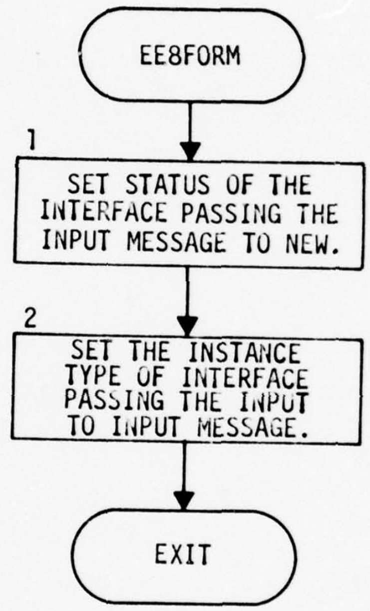



Figure 4-10 FORM (EE8FORM)

### 4.4.4 FOR EACH

Description

The FOR EACH module of the Data Manager is called during simulator execution by a user service request to gain access in a data-set or class to every instance or to every instance meeting some specified criterion. For each instance which is found (or found to meet the criterion), a section of user supplied code is executed. The user supplied code may include other Data Manager service requests such as FOR EACH, SELECT FIRST, DESTROY, etc. The FOR EACH criterion may be any valid Boolean expression but usually contains one or more DATA items of the data-set or class instance.

Input

Input to the FOR EACH module of the Data Manager are listed below:

a) An element of the list of data-sets/classes generated by the Data Translation module of the Simulator Generation function. The input element corresponds to a data-set (FILE or ENTITY_TYPE) or class (ENTITY_CLASS) and is an index to the desired entry in the Data-Set/Class Description Array. See Section 4.4 for a description of the Data-Set/Class Description Array.

b) An optional criterion (called a 'SUCH THAT' clause) used to qualify which instances are to be selected. Any Boolean expression is acceptable.

c) A section of user PDL 2 source code which may contain Data Manager service requests.

Processing

Processing performed in the FOR EACH module is shown in the flow diagram of Figure 4-11. The following comments refer to processing box numbers.

[1] - See SELECT FIRST module (Section 4.4.5) for detailed description. Inputs are inputs a) and b) for the FOR EACH module.

[2] - Test is made on global DATA item FOUND.

| | | |
|---|---|---|
| [3] | - | Currently-selected-instance-pointer within data-set is saved and the currently-selected-data-set within class is saved. |
| [4] | - | User supplied PDL 2 statements in an ALPHA (BETA or GAMMA) or SUBNET processing. |
| [5] | - | See UPDATE module for detailed description. Input is input a) for the FOR EACH module. The UPDATE module is called to save any newly created or selected instances of the FOR EACH input data-set or class. |
| [6] | - | Saved currently-selected-instance-pointer is stored in the saved currently-selected-data-set. The saved currently-selected-data-set is stored in its class currently-selected-data-set index. |
| [7] | - | See SELECT NEXT module for detailed description. Inputs are inputs a) and b) for the FOR EACH module. |

<u>Procedure Reference</u>

| | | |
|---|---|---|
| [1] | - | EE8UPDATE, EE8FIRST, EE8FOUND |
| [3] | - | EE8SAUDS |
| [5] | - | EE8UPDATE |
| [6] | - | EE8RESDS |
| [7] | - | EE8NEXT, EE8FOUND |

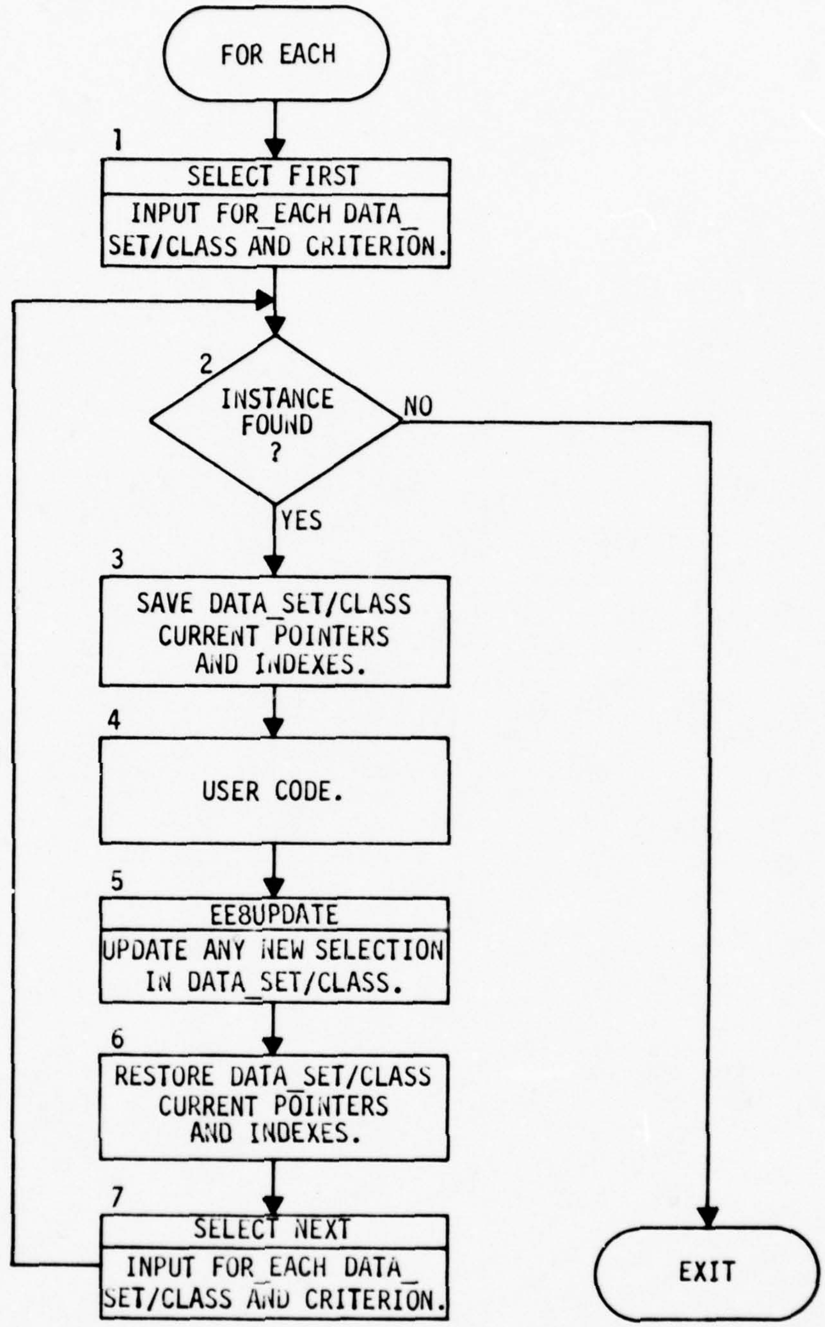



Figure 4-11 FOR EACH

### 4.4.5 SELECT FIRST

#### Description

The SELECT FIRST module of the Data Manager is called during simulator execution by a user service request to gain access in a data-set or class to the first instance (or to the first instance meeting some specified criterion). If an instance is not found (or not found to meet the criterion), the global DATA item FOUND is set to a value of FALSE, otherwise FOUND is set to TRUE. The criterion may be any valid Boolean expression but usually contains one or more DATA items of the data-set class instance.

#### Input

Inputs to the SELECT FIRST module of the Data Manager are listed below:

- An element of the list of data-sets/classes generated by the Data Translation module of the Simulator Generation function. The input element corresponds to a data-set (FILE or ENTITY_TYPE) or class (ENTITY_CLASS) and is an index to the desired entry in the Data-Set/Class Description Array. See Section 4.4 for a description of the Data-Set/Class Description Array.
- An optional criterion (called a 'SUCH THAT' clause) used to qualify which instances are to be selected. Any Boolean expression is acceptable.

#### Processing

Processing performed in the SELECT FIRST module is shown in the flow diagram of Figure 4-12. The following comments refer to processing box numbers.

[1] - See UPDATE module (Section 4.4.8) for detailed description.

[6, 15] - If the currently selected instance is marked as deleted and has a 'FOR EACH' examination count of zero, it will be delinked from its owning list and disposed.

[16] - This decision is implemented in the user's BETA or GAMMA by the presence (YES) or absence (NO) of the source code to perform the operations [17] through [23].

| | | |
|---|---|---|
| [17-23] | - | This is an in-line 'while loop' which calls EE8NEXT until the 'SUCH THAT' clause is satisfied or the end of the data-set or class is reached. |

Procedure Reference

| | | |
|---|---|---|
| [1] | - | EE8UPDATE |
| [2-15] | - | EE8FIRST |
| [24-31] | - | EE8FOUND |

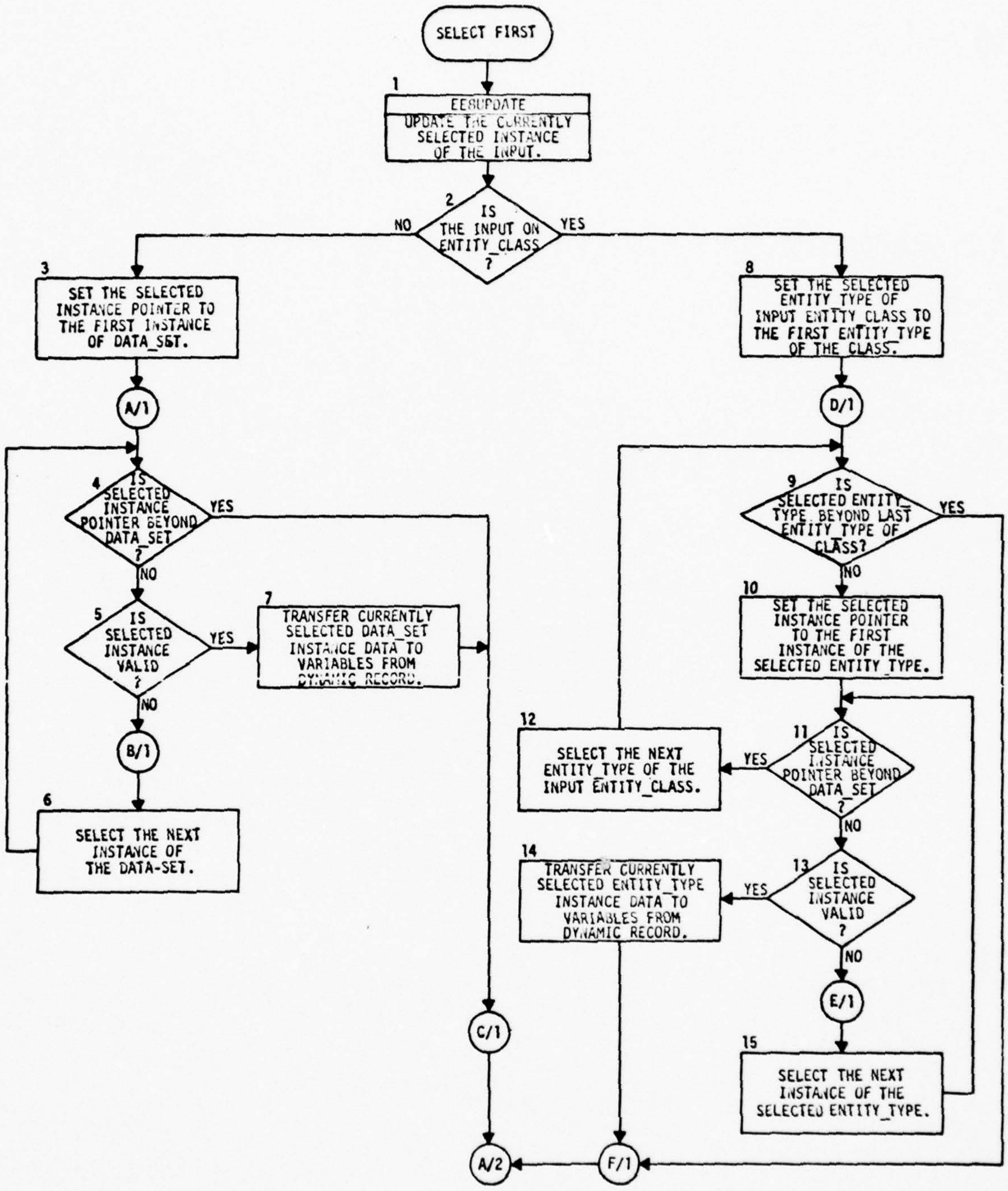



Figure 4-12 SELECT FIRST

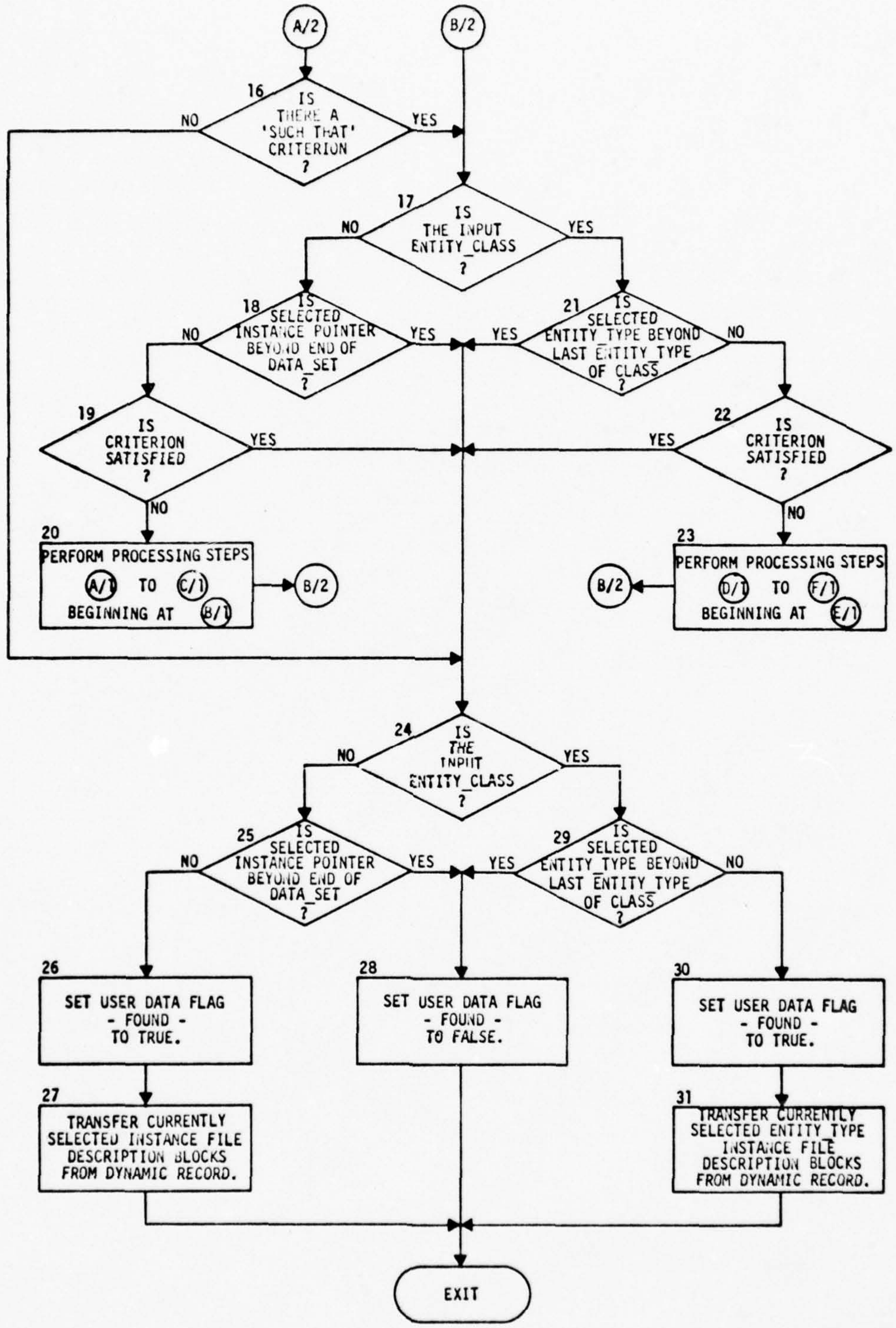



Figure 4-12 SELECT FIRST (Continued)

### 4.4.6 SELECT NEXT

Description

The SELECT NEXT module of the Data Manager is called during simulator execution by a user service request to gain access in a data-set or class to the next instance (or to the next instance meeting some specified criterion). If an instance is not found (or not found to meet the criterion), the global DATA item FOUND is set to a value of FALSE, otherwise FOUND is set to TRUE. The criterion may be any valid Boolean expression but usually contains one or more DATA items of the data-set/class instance. If no valid instance is currently selected when the SELECT NEXT module is invoked, a diagnostic message will be issued and no operation performed.

Input

Input to the SELECT NEXT module of the Data Manager are listed below:

- An element of the list of data-sets/classes generated by the Data Translation module of the Simulation Generation function. The input element corresponds to a data-set (FILE or ENTITY CLASS) and is an index to the desired entry in the Data-Set/Class Description Array. See Section 4.4 for a description of the Data-Set/Class Description Array.

- An optional criterion (called a 'SUCH THAT' clause) used to qualify which instances are to be selected. Any Boolean expression is acceptable.

Processing

Processing performed in the SELECT NEXT module is shown in the flow diagram of Figure 4-13. The following comments refer to processing box numbers.

| | | |
|---|---|---|
| [1] | - | See UPDATE module for detailed description. |
| [7, 17] | - | If the currently selected instance is marked as deleted and has a 'FOR EACH' examination count of zero, it will be delinked from its owning list and disposed. |

| | | |
|---|---|---|
| [18] | - | This decision is implemented in the users BETA or GAMMA by the presence (YES branch) or absence (NO branch) of the source code to perform the operations boxes 19 through 25. |
| [19-25] | - | This is an in line 'while loop' which calls EE8NEXT until the 'SUCH THAT' clause is satisfied or the end of the data-set or class is reached. |

Procedure Reference

| | | |
|---|---|---|
| [1] | - | EE8UPDATE |
| [2-17] | - | EE8NEXT |
| [26-33] | - | EE8FOUND |

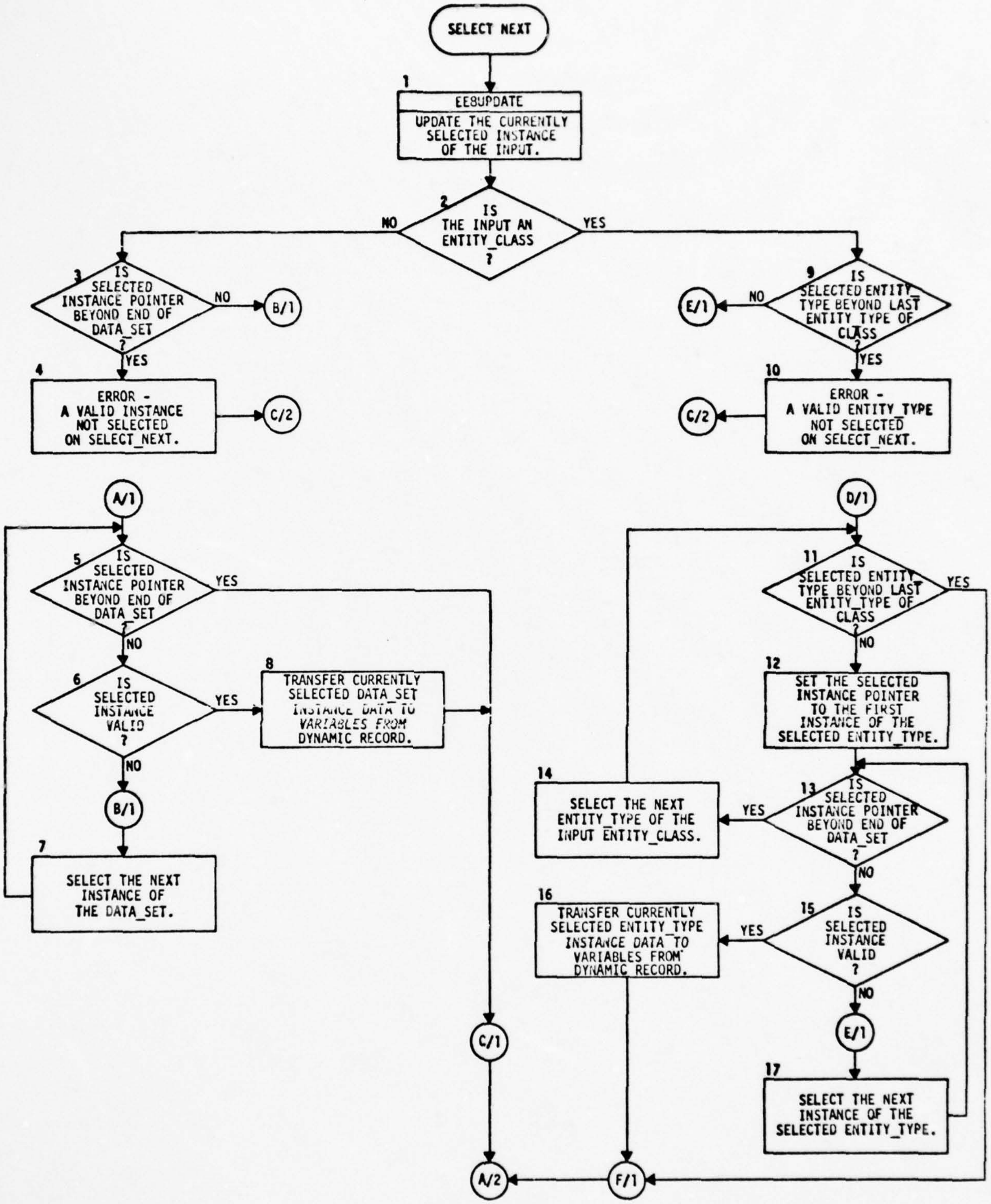



Figure 4-13 SELECT NEXT

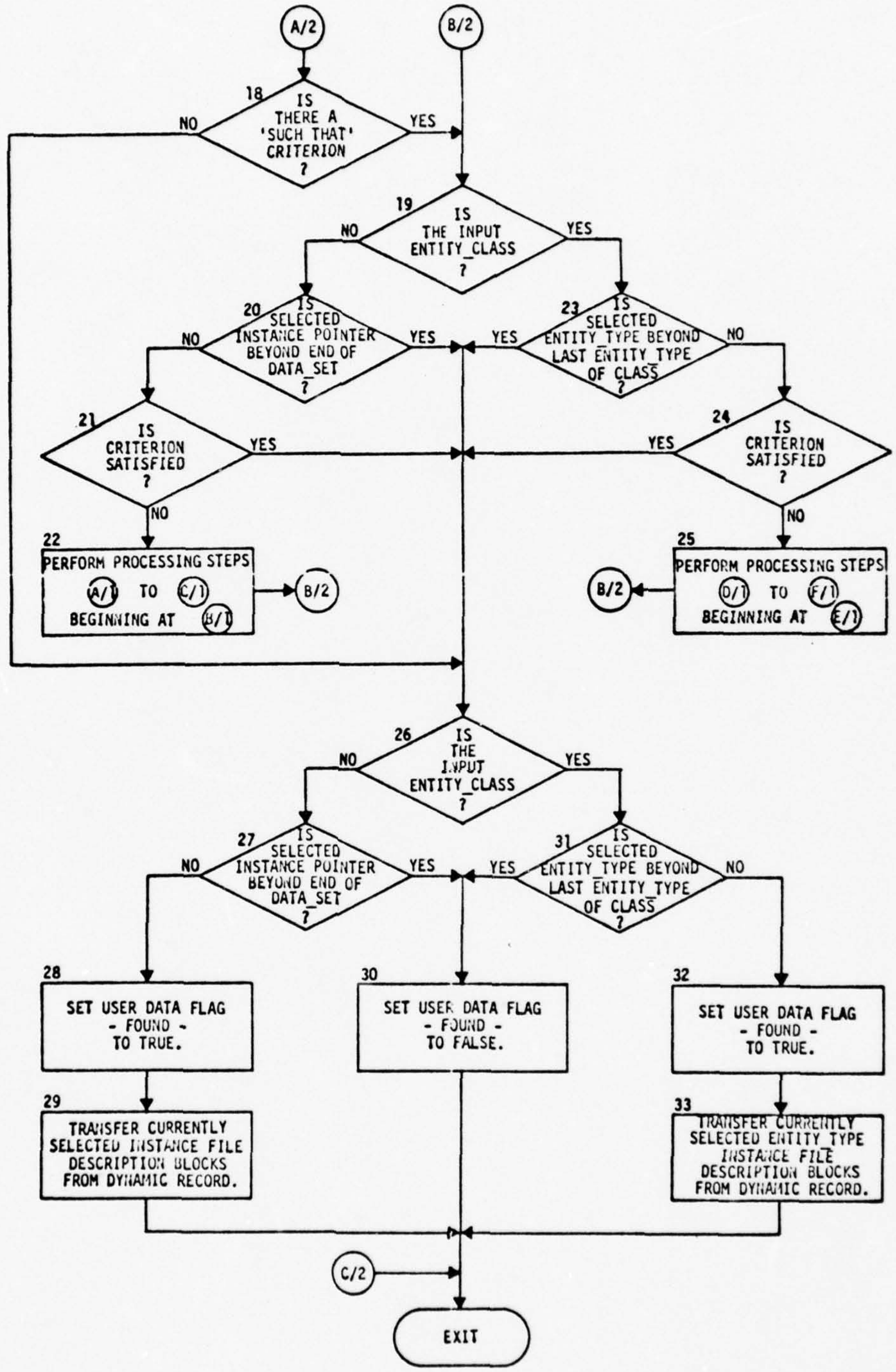



Figure 4-13 SELECT NEXT (Continued)

### 4.4.7 SET (EE8SETYP)

#### Description

The SET module of the Data Manager is called during simulator execution by a user service request to establish the type of a previously selected or newly created ENTITY_CLASS. When a user creates an instance of an ENTITY_CLASS, no type is specified and no ENTITY_TYPE FILEs or DATA values can be created -- only ENTITY_CLASS FILEs and DATA values. The user must perform a SET service request to establish the ENTITY_TYPE of the instance before any ENTITY_TYPE FILEs and DATA values can be created. If the user fails to SET ENTITY_TYPE before the next select or create operation on the ENTITY_CLASS, a diagnostic message is issued and the instance is destroyed. A SET service request may be used to transform a selected ENTITY_TYPE instance to an instance of another ENTITY_TYPE in the same ENTITY_CLASS. All ENTITY_CLASS and common FILEs and DATA will be preserved while all non-common FILEs and DATA will be initialized.

#### Input

Input to the SET module of the Data Manager is an element of the list of data-sets/classes. The input element corresponds to an ENTITY_TYPE and indexes the desired data-set entry in the Data-Set/Class Description Array. See Section 4.4 for a description of the Data-Set/Class Description Array.

#### Processing

Processing performed in the SET module is shown in the flow diagram of Figure 4-14. The following comments refer to processing box numbers in the diagram.

[2, 3] - An ENTITY_CLASS with no ENTITY_TYPE selected and with a true value for new status has a newly created instance which has no ENTITY_TYPE yet assigned it.

[8, 9, 27] - The diagnostic message is written to external file OUTPUT.

[9] - A valid instance may be described as in existence (currently-selected-instance pointer is not NIL) and not marked as deleted.

Procedure Reference

| | | |
|---|---|---|
| [7] | - | EE8ININ |
| [8, 10, 27] | - | EE8LSTDIAG |
| [12, 18] | - | EE8XIO |
| [13, 19] | - | EE8XFO |
| [14, 20] | - | EE8ININ |
| [15, 21] | - | EE8XII |
| [16, 22] | - | EE8XFI |
| [17] | - | EE8NEWI |
| [23] | - | EE8DISI |

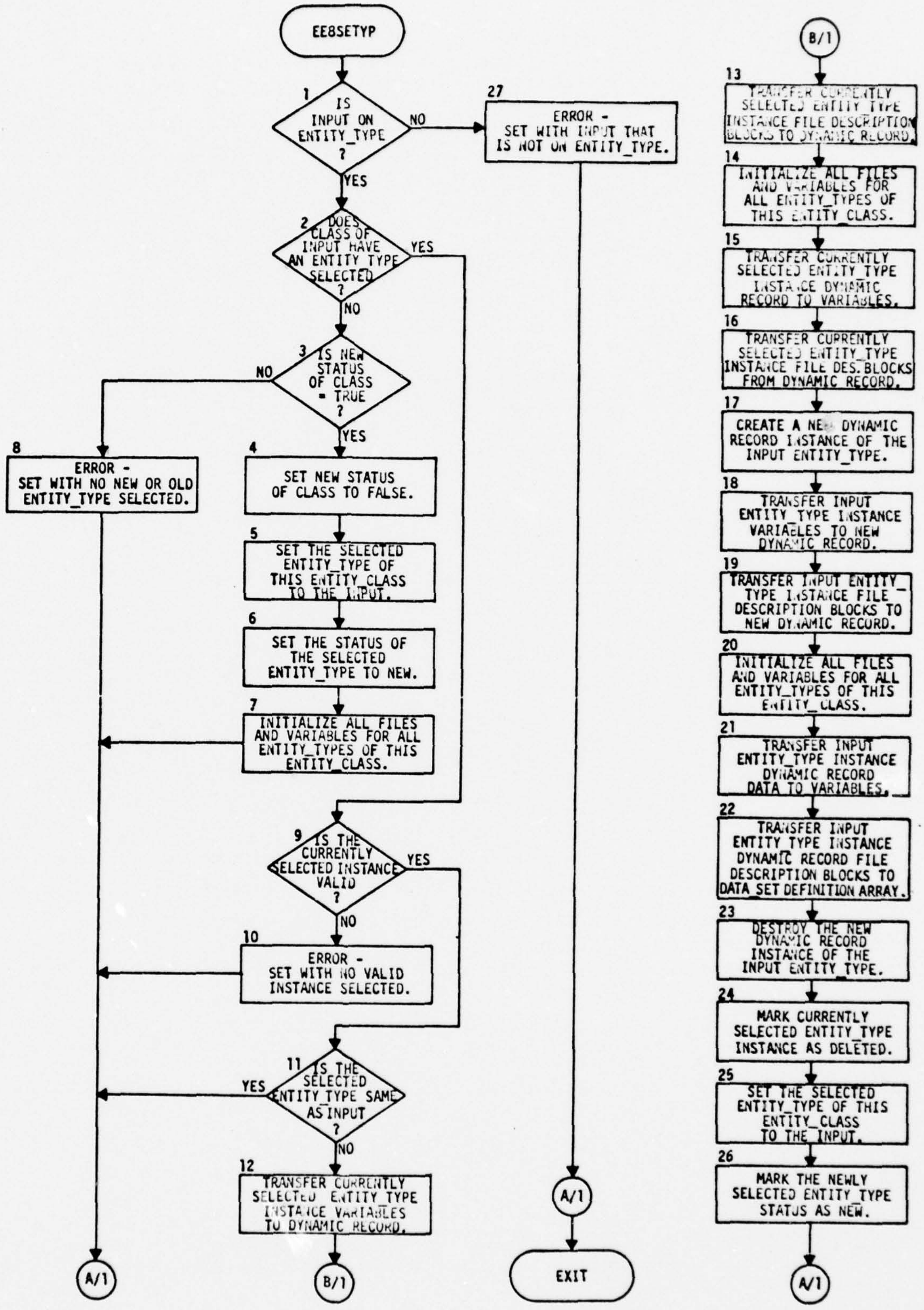



Figure 4-14 SET (EE8SETYP)

### 4.4.8 UPDATE (EE8UPDATE)

Description

The UPDATE module of the Data Manager is called during simulator execution by other Data Manager modules to store modified data values from the static variables of an instance into a dynamic record linked on an ordered list (either FIFO or Ranked High). This operation is done to prepare for the next operation on this data-set which will select or create a new instance of the data-set. In the case of a Ranked High data-set in which the ordering variable has been changed in such a way as to force the current instance into a new position, a new dynamic record must be created and linked into the new position before the data values can be moved. The old instance will be marked for deletion. For a newly created instance, a new dynamic record must also be created and linked into its proper position (based on the ranking scheme for this data-set) before the data can be transferred into the dynamic record.

Input

Input to the UPDATE module of the Data Manager is an element of the list of data-sets/classes. The input element acts as an index to the desired data-set (FILE or ENTITY_TYPE) or class (ENTITY_CLASS) entry in the Data-Set/Class Description Array. See Section 4.4 for a description of the Data-Set/Class Description Array.

Processing

Processing performed in the UPDATE module is shown in the flow diagram of Figure 4-15. The following comments refer to processing box numbers.

| | | |
|---|---|---|
| [15, 35, 39] | - | Diagnostic message is written to external file OUTPUT. |
| [36] | - | Processing is done on the selected ENTITY_TYPE of the ENTITY_CLASS to which the input ENTITY_TYPE belongs. In other words, the processed ENTITY_TYPE is not necessarily the input ENTITY_TYPE. |

| | | |
|---|---|---|
| [14] | - | A valid instance may be described as in existence (currently-selected-instance pointer is non-NIL) and not marked as deleted. |
| [26] | - | If the saved-current-instance is the same as the present-current-instance, the instance has not changed position in the linked list even though the ordering variable has changed value. |
| [9, 25] | - | If the currently-selected-instance is marked as deleted and has a 'FOR EACH' examination count of zero, it will be delinked from its owning list and disposed. |

<u>Procedure Reference</u>

| | | |
|---|---|---|
| [35, 39] | - | EE8LSTDIAG |
| [3-32] | - | EE8UDSS |

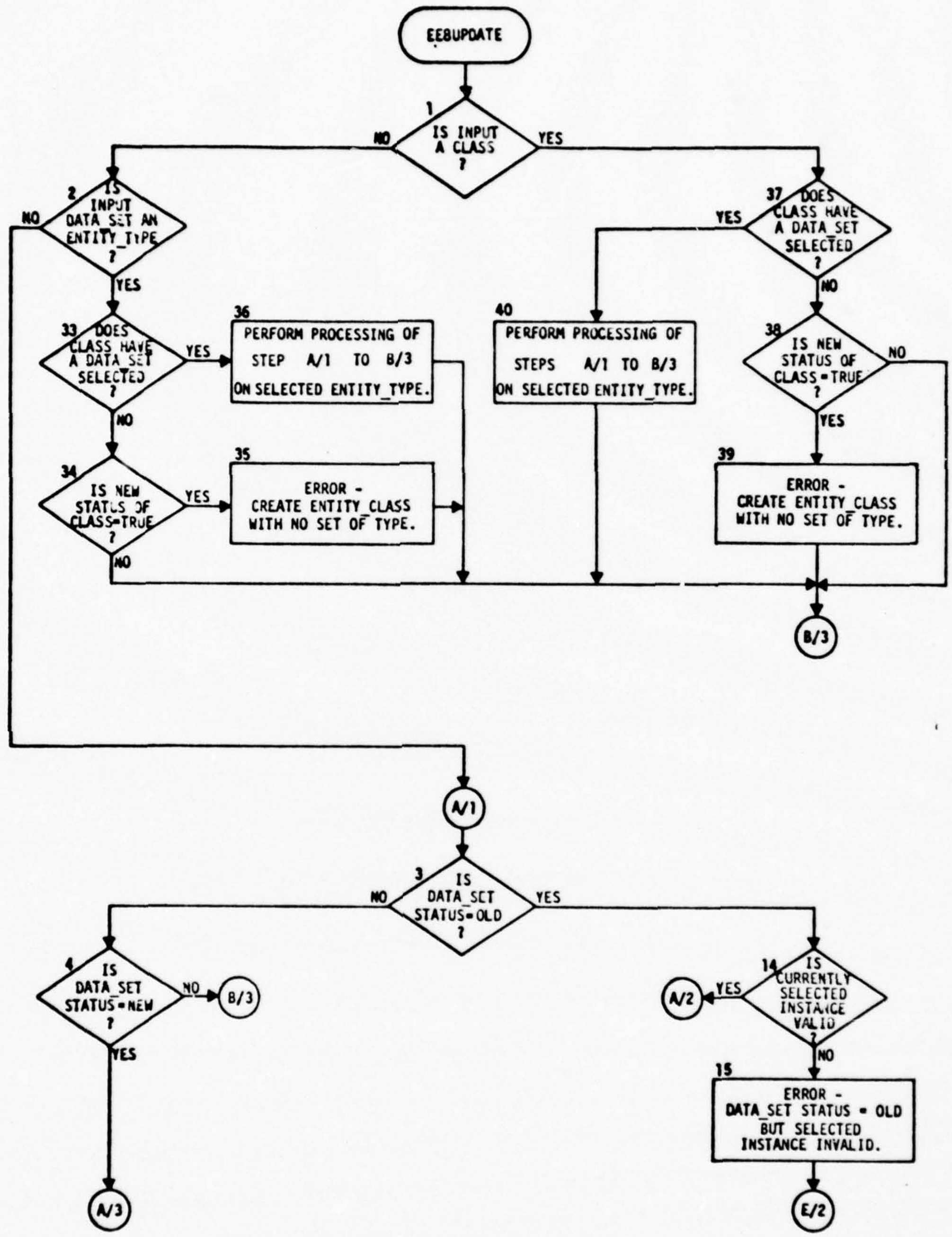



Figure 4-15 UPDATE (EE8UPDATE)

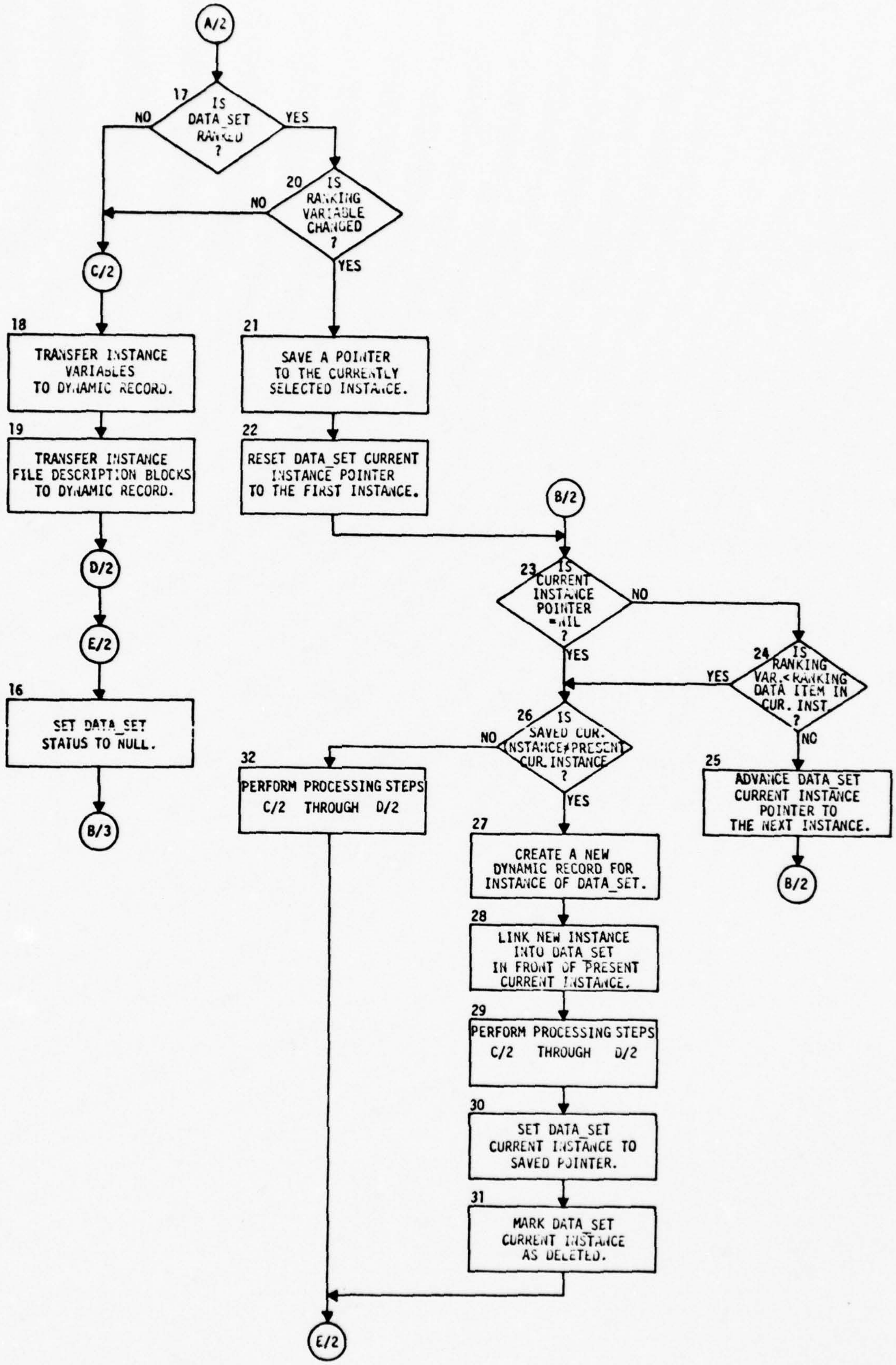



Figure 4-15 UPDATE (EE8UPDATE) (Continued)

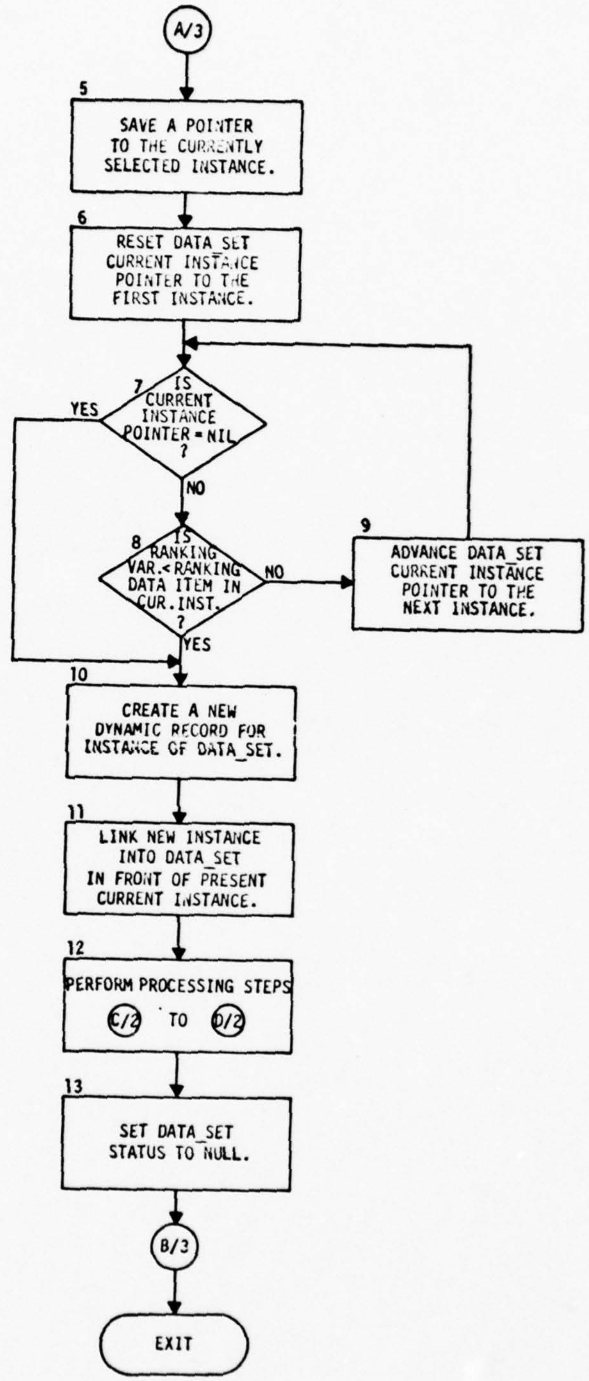



Figure 4-15 UPDATE (EE8UPDATE) (Continued)

## 5.0 COMPUTER PROGRAM DESCRIPTION - REVS GENERATED SIMULATOR POST-PROCESSOR PROGRAM

### Description

The Simulator Post-Processor Program (VVEXEC) is a PDL2 main program constructed by the Simulator Generation function. Once the Simulator Post-Processor Program is constructed and the necessary user inputs are supplied through the Simulator Post-Processor Execution (SIMDA), the program is completely executable outside the control of REVS. The Simulator Post-Processor Program is composed of the following major processing elements as shown in Figure 5-1.

- TEST Procedures
- Simulator Post-Processor Initialization
- Simulator Post-Processor Executive
- Simulator Post-Processor Data Management

The Simulator Post-Processor Program is designed to provide an environment for the selective execution of PERFORMANCE_REQUIREMENT TESTs allowing access to all DATA and FILES recorded at VALIDATION_POINTS during the Simulator Execution.

### Input

| | | |
|---|---|---|
| VALIDATION INFORMATION FILE (VVIF) | - | Text file of user requests of tests to be executed. |
| RECORDING DATABASE | - | A DBCS database of all DATA and FILES recorded at VALIDATION_POINTS during an execution of the simulator. |

### Output

| | | |
|---|---|---|
| TEST RESULT FILE (OUTPUT) | - | Text file indicating whether an executed test passed or failed. |
| DIAGNOSTIC FILE (OUTPUT) | - | A text file informing the user of any errors detected during the Simulator Post-Processor Program. |

Processing

The control flow through the Simulator Program is shown in Figure 5-2. As shown in the flow diagram, control is transferred first to Simulator Post-Processor Initialization (VVINITL) and then to the Simulator Executive (VVMAIN) which controls the execution of TESTS depending upon user requests. These two functions as well as the Simulator Post-Processor Data Manager are described in the following sections.

Procedure Reference

The processing depicted in the blocks of Figure 5-2 is performed by the following software procedures in the Simulator Post-Processor Program.

[1] - VVINITL

[2] - VVMAIN

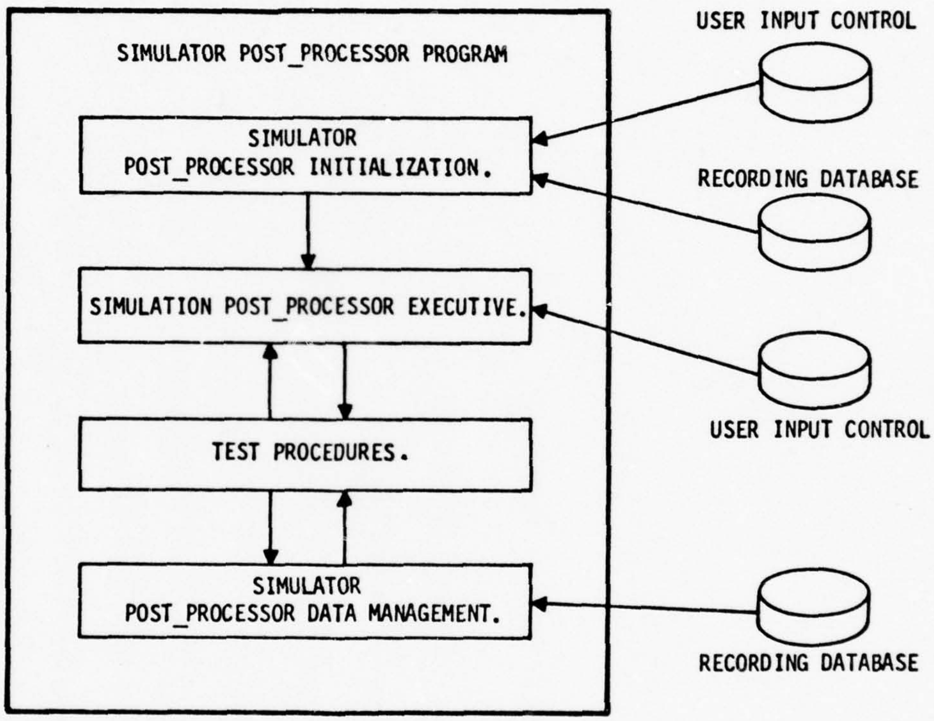



Figure 5-1 Simulator Post-Processor Overview

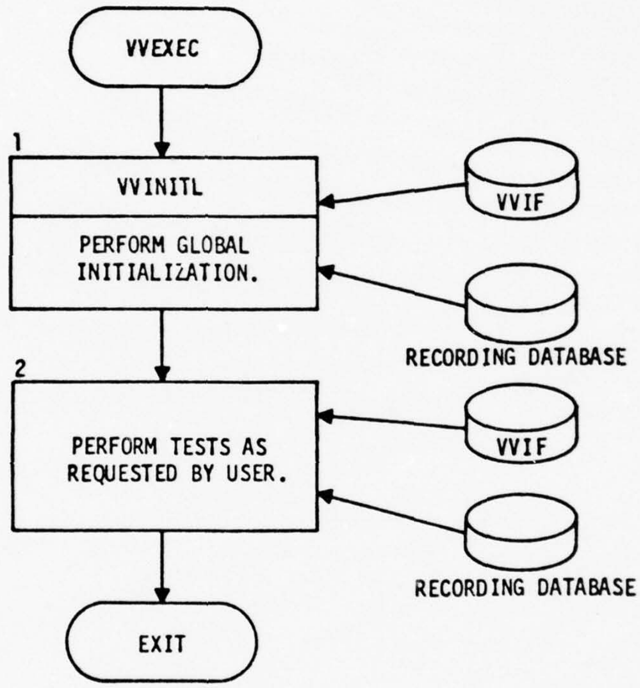



Figure 5-2 Simulator Post-Processor Program (VVEXEC)

## 5.1 SIMULATOR POST-PROCESSOR INITIALIZATION (VVINITL)

### Description

The purpose of the Simulator Post-Processor Initialization Function is to perform all initialization of variables and arrays required for the execution of the Simulator Post-Processor Program. In addition, the Simulator Post-Processor Initialization Function insures that program inputs are matched to the program. This means that the recording database is generated by the Simulator which was built at the same time as the Simulator Post-Processor and that the Validation Input File was generated by a Simulator Post-Processor Execution request for this Simulator.

### Input

| | | |
|---|---|---|
| VALIDATION INFORMATION FILE (VVIF) | - | Text file of user requests for tests to be executed. The Simulator Post-Processor Initialization Function accesses only the identification information on this file. It includes the date and time of Simulator creation and the user assigned Simulator ID |
| RECORDING DATABASE | - | A DBCS database of all DATA and FILES recorded at VALIDATION_POINTS during an execution of the simulator. The simulator Post-Processor Initialization Function accesses the identification information (Simulator creation time and date, Simulator user assigned ID) and the VALIDATION_POINTS. |

### Output

| | | |
|---|---|---|
| DIAGNOSTIC FILE (OUTPUT) | - | A text file informing the user of any errors detected during the Simulator Post-Processor Initialization Function. |
| CONTENTS OF THE VALIDATION_POINT ARRAY (VV9VP) INITIALIZED | - | An array of records each of which give the DBCS address of a VALIDATION_POINT and the DBCS address of the currently selected RECORDING of the VALIDATION_POINT. |
| CONTENTS OF THE FILE ARRAY (VV9FL) INITIALIZED | - | An array of records each of which give the DBCS address of a VALIDATION_POINT FILE and the DBCS address of the currently selected RECORD of the VALIDATION_POINT FILE. |

Processing

The control flow through the Simulator Post-Processor Initialization Function is shown in Figure 5-3. The following comments refer to processing steps shown in the flow diagram.

| | | |
|---|---|---|
| [3] | - | Matching IDs include creation date, creation time, and user supplied ID. |
| [7] | - | Same as [3] above. |

Procedure References

The following correlates the functional processing elements shown in Figure 5-3 with the Simulator Post-Processor Initialization procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1-16] | - | VVINITL |

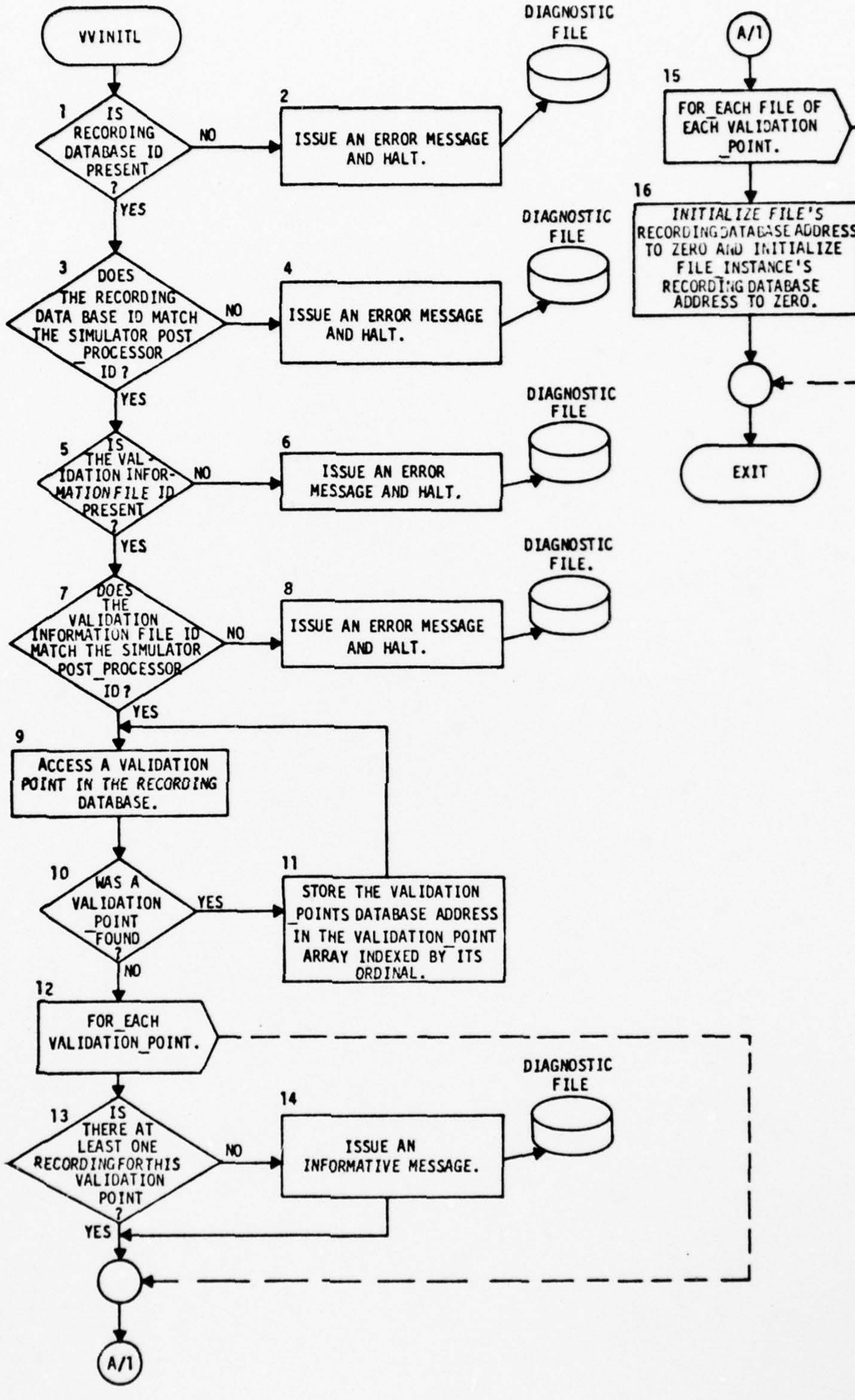



Figure 5-3 Simulator Post-Processor Initialization (VVINITL)

## 5.2 SIMULATOR POST-PROCESSOR EXECUTIVE (VVMAIN)

Description

The purpose of the Simulator Post-Processor Executive Function is to invoke the executable description of PERFORMANCE_REQUIREMENTS on a selective basis. The user TEST requests are read from the Validation Information File and acted on as they are read. If the user requests a TEST which is not in the Simulator Post-Processor Program, a diagnostic is issued informing him of his error.

Input

| | | |
|---|---|---|
| VALIDATION INFORMATION FILE | - | Text file of user requests of tests to be executed. |

Output

| | | |
|---|---|---|
| TEST RESULT FILE (OUTPUT) | - | Text file indicating whether an executed test passed or failed. |
| DIAGNOSTIC FILE (OUTPUT) | - | Text file informing the user of TESTs which could not be located. |

Processing

The control flow diagram through the Simulator Post-Processor Executive Function is shown in Figure 5-4. The following comments refer to processing steps shown in the flow diagram.

| | | |
|---|---|---|
| [5] | - | Results are PASS or FAIL only. |
| [6] | - | If all TESTs have been requested, no messages are issued for TESTs not found. |

Procedure References

The following correlates the functional processing elements shown in Figure 5-4 with the Simulator Post-Processor Executive procedures which perform the indicated processing.

| | | |
|---|---|---|
| [1] | - | VV8RFVV |
| [2-4] | - | VVMAIN |
| [5] | - | VV8PRPF |
| [6-7] | - | VVMAIN |

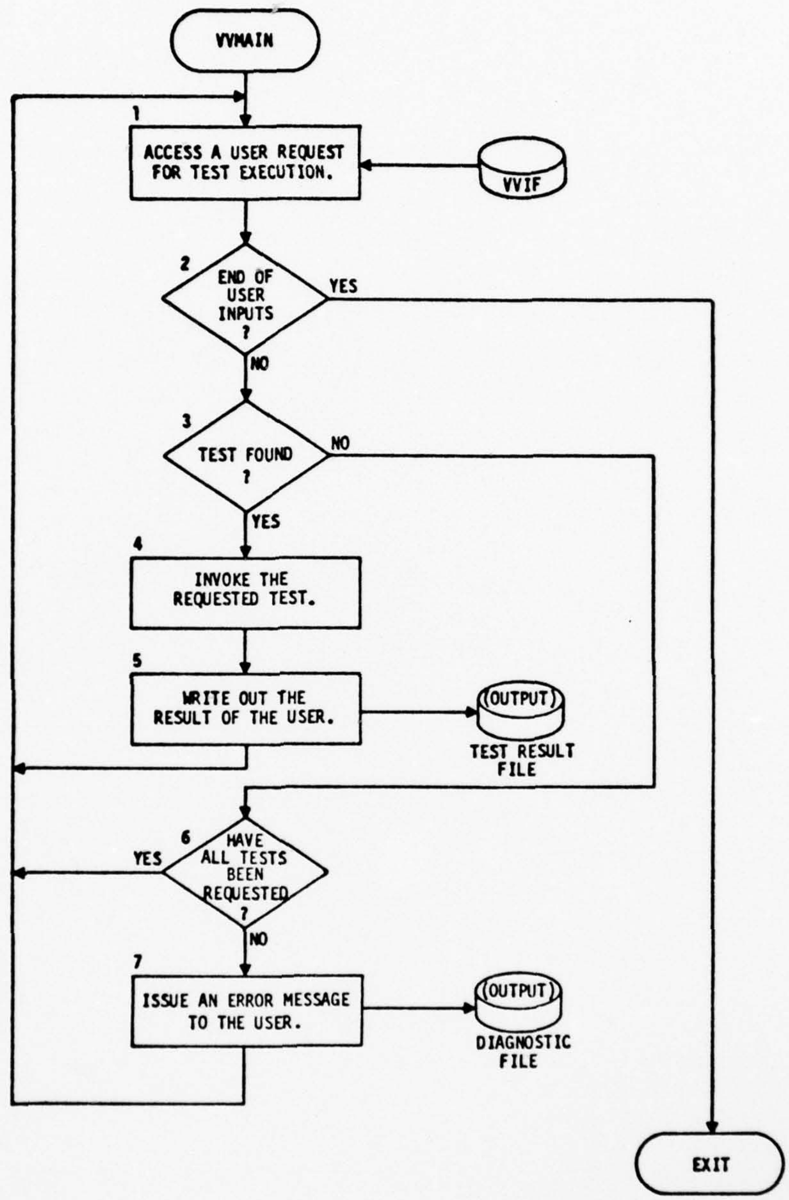



Figure 5-4 Simulator Post-Processor Executive (VVMAIN)

## 5.3 SIMULATOR POST PROCESSOR DATA MANAGEMENT

The Simulator Post-Processor Data Manager provides the services requested in the executable description (TEST) of PERFORMANCE_REQUIREMENTs. Specifically, it supports RECORDING and RECORD access using a DBCS database system.

The service requests supported are:

- access first or next instance of a VALIDATION_POINT's RECORDINGs.
- access all instances of a VALIDATION_POINT's RECORDINGs.
- access first or next instance of a FILE (RECORDED at a VALIDATION_POINT).
- access all instances of a FILE (RECORDED at a VALIDATION_POINT).

The Simulator Data Manager is actually a set of procedures which implement the required service requests by calling DBCS routines to manipulate the recording database. Each VALIDATION_POINT is assigned an entry in the VALIDATION_POINT array, VV9VP, and each VALIDATION_POINT FILE is assigned an entry in the FILE array, VV9FL. The entries of both these arrays contain the same basic data -- first is the DBCS address of the FILE or the VALIDATION_POINT and second is the DBCS address of the current instance (RECORD/RECORDING) of the FILE or VALIDATION_POINT. Data Management service calls are translated into procedure invocations which pass a control parameter to indicate what type of operation is requested (FIRST or NEXT). A unique procedure is invoked for each VALIDATION_POINT and each FILE.

A user can only access a DATA value or select a FILE after a successful selection of a VALIDATION_POINT RECORDING. Similarly, FILE DATA can only be accessed after successful selection of a FILE. All DATA is qualified by the VALIDATION_POINT name.

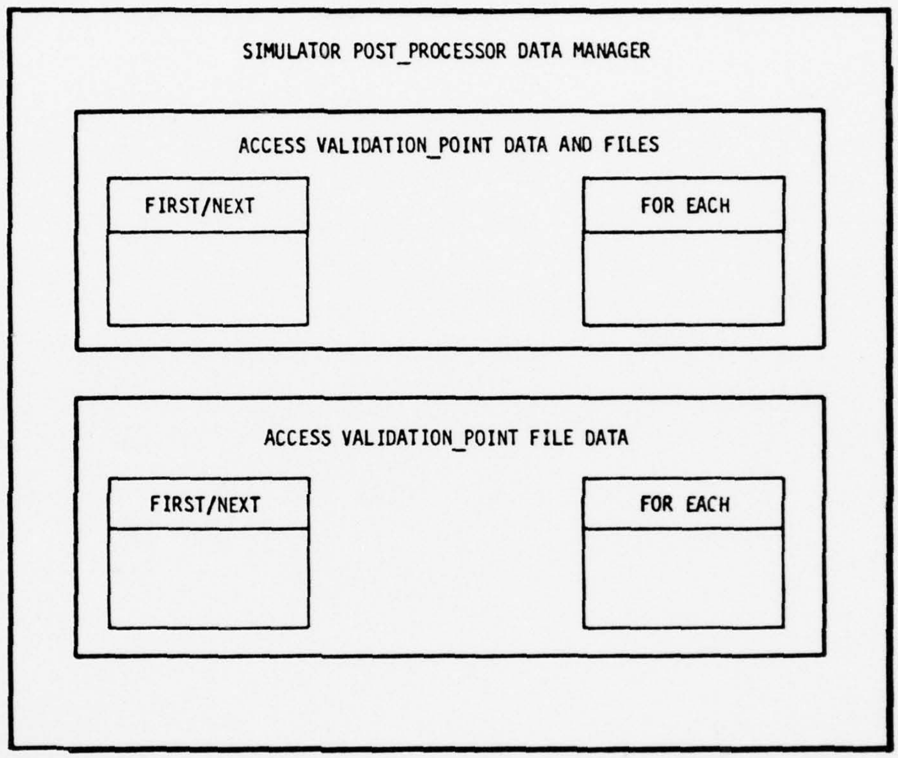



Figure 5-5 Simulator Post-Processor Data Manager Components

### 5.3.1 SELECT FIRST/NEXT VALIDATION_POINT RECORDING (SFNVPR)

Description

The SELECT FIRST/NEXT VALIDATION_POINT RECORDING (SFNVPR) module of the Data Manager is called during Simulator Post Processor execution by a user service request to gain access in a VALIDATION_POINT to the first/next RECORDING (or to the first/next RECORDING meeting some user specified criterion). If such a RECORDING is not found, the unqualified DATA item RECORDING_FOUND is set to a value of FALSE, otherwise RECORDING_FOUND is set to TRUE. The user specified criterion may be any valid Boolean expression but usually would contain one or more DATA items RECORDED at the VALIDATION_POINT. Note that all DATA items RECORDED at a VALIDATION_POINT, are declared as PDL 2 fields of a record with the same name as the VALIDATION_POINT (thus the user must reference DATA item X at VALIDATION_POINT Y as Y.X).

Input

Inputs to the SELECT FIRST/NEXT VALIDATION_POINT RECORDING module are listed below:

- A parameter specifying first or next RECORDING.
- An optional criterion (called a 'SUCH THAT' clause) used to qualify which RECORDINGs are to be selected. Any boolean expression is acceptable.

Processing

Processing performed in SFNVPR module is shown in the flow diagram of Figure 5-6. The following comments refer to processing box numbers:

| | | |
|---|---|---|
| [1] | - | A DBCS address of zero for a VALIDATION_POINT implies that there are no RECORDINGs for it in the Recording Database. |
| [5] | - | This is implemented by one DBCS procedure invocation per DATA item read in. |
| [6] | - | This decision is implemented in the user's TEST by the presence (YES) or absence (NO) of the code to perform the operations [7-9]. |
| [7-9] | - | This is an in-line 'while loop' which invokes the SFNVPR module with input parameter 'NEXT' until the end of RECORDINGs or the 'SUCH THAT' clause is satisfied. |

Procedure Reference

[1-5] - VVVXXXX

[10-13] - VV8FOUND

Note that XXXX is the VALIDATION_POINT ordinal assigned by the VALIDATION_POINT Translation Module of the SIMGEN function. There is one procedure (with name of the form VVVXXXX) for each VALIDATION_POINT corresponding to processing steps [1] through [5].

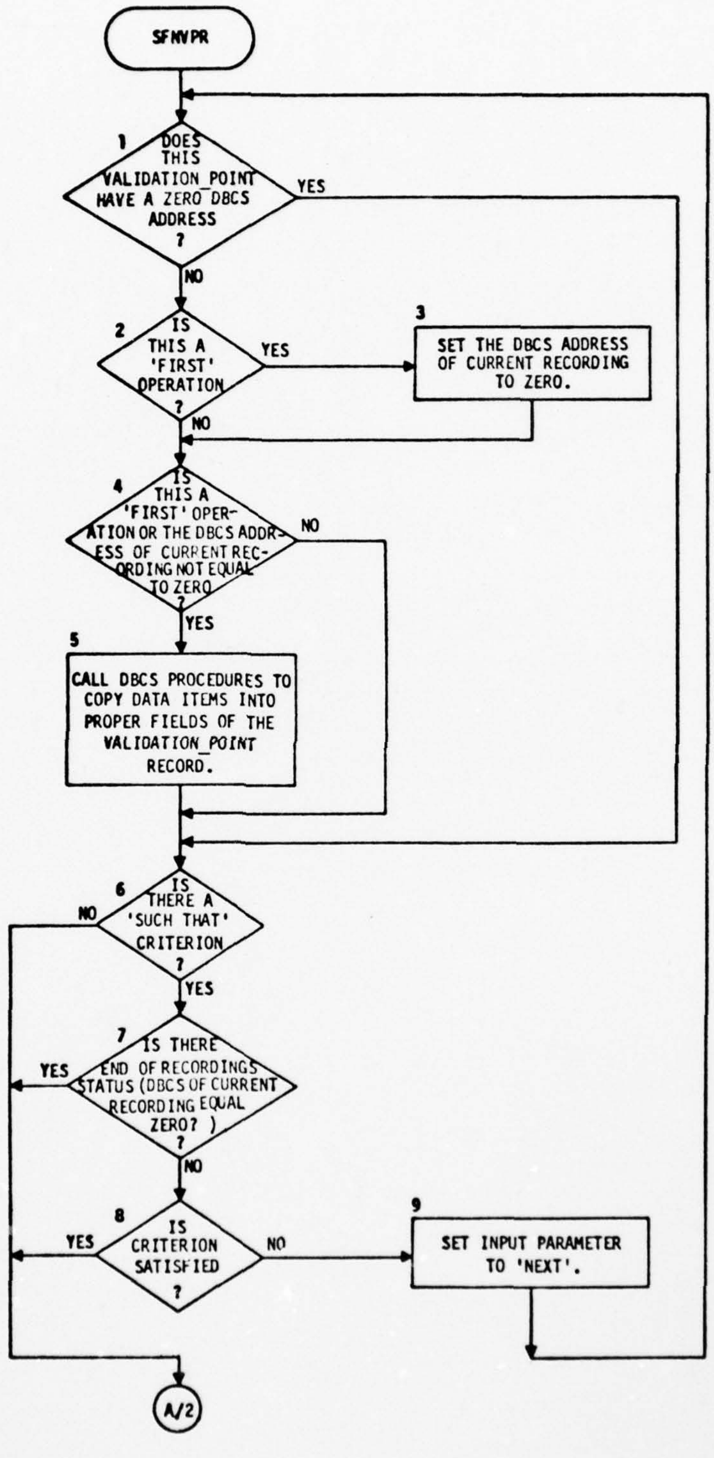



Figure 5-6 Select First/Next Validation-Point Recording (SFNVPR)

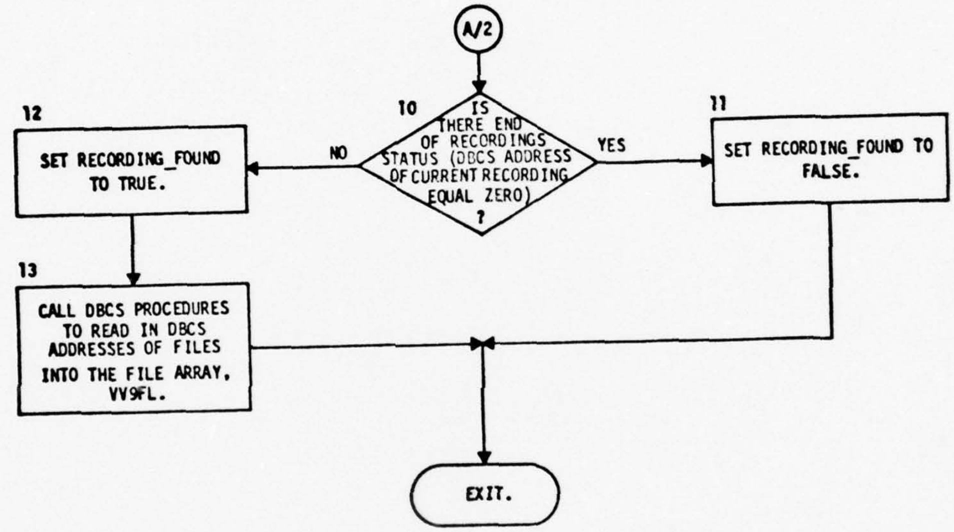



Figure 5-6 Select First/Next Validation-Point Recording (SFNVPR) (Continued)

### 5.3.2 FOR EACH VALIDATION_POINT RECORDING (FEVPR)

Description

The FOR EACH VALIDATION_POINT RECORDING (FEVPR) module of the Data Manager is called during Simulator Post Processor execution by a user service request to gain access in a VALIDATION_POINT to every RECORDING, or to every RECORDING meeting some user specified criterion. For each RECORDING which is found (or found to meet the criterion), a section of user supplied code is executed. The user supplied code may include other Data Manager service requests such as SELECT, FOR EACH, etc. The FOR EACH criterion may be any valid boolean expression but usually would contain one or more DATA items RECORDED at a VALIDATION_POINT. All DATA items RECORDED at a VALIDATION_POINT must be referenced as "VALIDATION_POINT name. DATA item name".

Input

Inputs to the FOR EACH VALIDATION_POINT RECORDING module are listed below:

- An optional criterion (called a 'SUCH THAT' clause) used to qualify which RECORDINGs are to be selected. Any boolean expression is acceptable.
- A section of user PDL 2 source statements which may contain Data Manager service requests.

Processing

Processing performed in the FEVPR module is shown in the flow diagram of Figure 5-7. The following comments refer to processing box numbers.

[1],[6] - See the SELECT FIRST/NEXT VALIDATION POINT RECORDING module (Section 5.3.T) for a detailed description.

[2] - Test is made on unqualified DATA item, RECORDING_FOUND.

Procedure Reference

In-line code is generated except for calls to the SELECT FIRST/NEXT VALIDATION_POINT RECORDING module which are of the form VVVXXXX (XXXX is the ordinal of the VALIDATION_POINT).

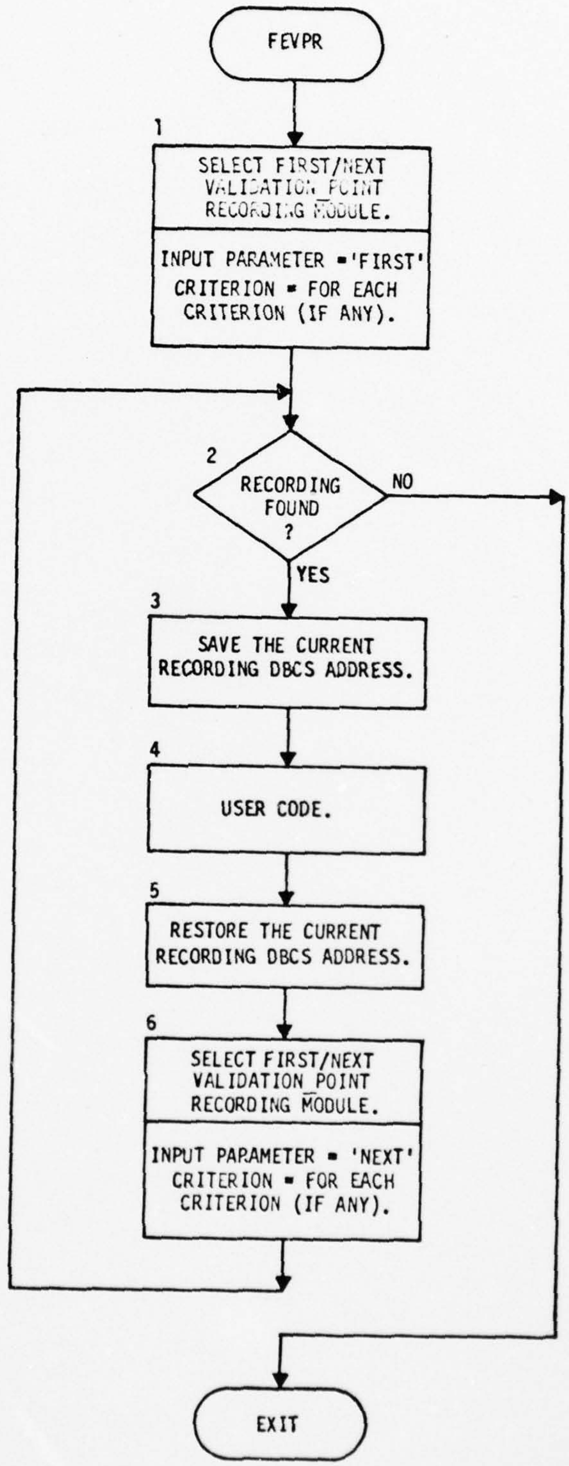



Figure 5-7 For Each Validation-Point Recording (FEVPR)

5.3.3 SELECT FIRST/NEXT FILE RECORD (SFNFR)

Description

The SELECT FIRST/NEXT FILE RECORD (SFNFR) module of the Data Manager is called during Simulator Post Processor execution by a user service request to gain access in a VALIDATION_POINT FILE to the first/next instance (RECORD) or to the first/next instance meeting some user specified criterion. If such a RECORD is not found, the unqualified DATA item RECORD_FOUND is set to a value of FALSE, otherwise RECORD_FOUND is set to true. The user specified criterion may be any valid Boolean expression but usually would contain one or more DATA items in the FILE. Note that a VALIDATION_POINT FILE cannot be SELECTed until a RECORDING of the VALIDATION_POINT has been SELECTed. All DATA items including those of FILEs must be qualified by the VALIDATION_POINT name at which they were RECORDed.

Inputs

Inputs to the SELECT FIRST/NEXT FILE RECORD module are listed below:

- A parameter specifying 'FIRST' or 'NEXT' RECORD.
- An optional criterion (called a 'SUCH THAT' clause) used to qualify which RECORDs are to be selected. Any boolean expression is acceptable.

Processing

Processing performed in the SELECT FIRST/NEXT FILE RECORD module is shown in the flow diagram of Figure 5-8. The following comments refer to processing box numbers.

| | | |
|---|---|---|
| [1] | - | A DBCS Address of zero for a FILE implies that there are no RECORDs for it in the Recording Database. |
| [5] | - | This is implemented by one DBCS procedure invocation per DATA item. |
| [6] | - | This decision is implemented in the user's TEST by the presence (YES) or absence (NO) of the code to the operations [7-9]. |
| [7-9] | - | This is implemented as an in-line 'while loop' which invokes the SELECT FIRST/NEXT RECORD module with input parameter 'NEXT' until the end of RECORDs or the 'SUCH THAT' clause is satisfied. |

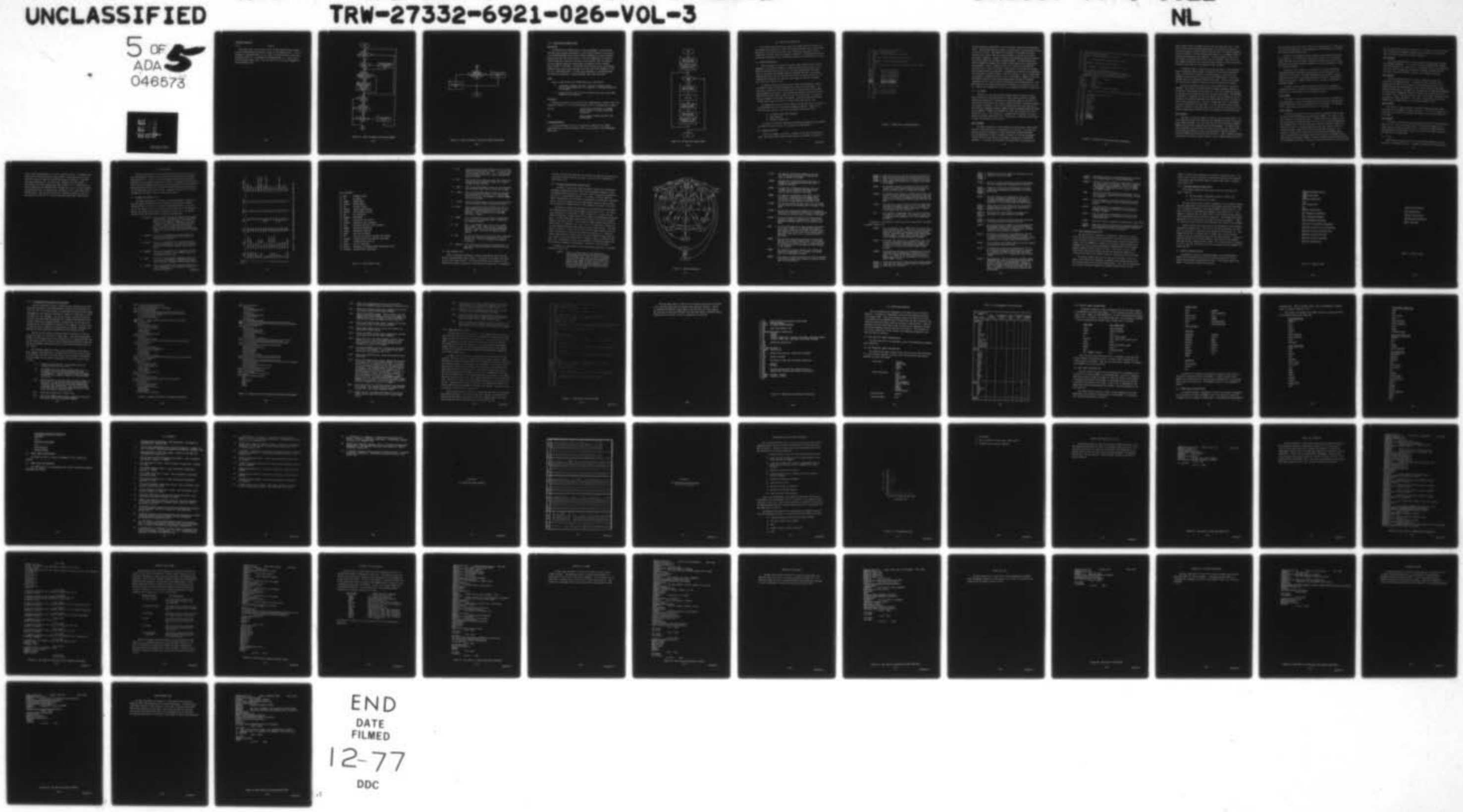
AD-A046 573    TRW DEFENSE AND SPACE SYSTEMS GROUP HUNTSVILLE ALA    F/G 9/2

REVS MAINTENANCE MANUAL. SREP FINAL REPORT. VOLUME III.(U)

AUG 77  W E BENOIT, P N BERGSTRESSER    DASG60-75-C-0022

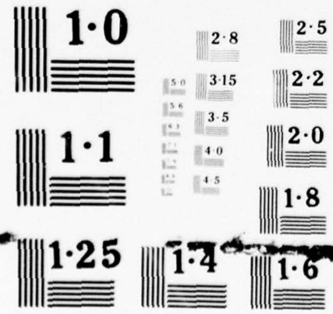
UNCLASSIFIED    TRW-27332-6921-026-VOL-3    NL

5 OF 5
ADA
046573



END
DATE
FILMED
12-77
DDC

04657




NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

Procedure Reference

[1-5] - VVYYYYY

Note that YYYYY is the ordinal of the VALIDATION_POINT FILE assigned by the VALIDATION_POINT Translation module of the SIMGEN function. FILE 'A' RECORDED at VALIDATION_POINT 'B' will have a different ordinal (and access procedure) from FILE 'A' RECORDED at VALIDATION_POINT 'C'. There is one access procedure (with name of the form VVYYYYY) for each FILE RECORDED at each VALIDATION_POINT.

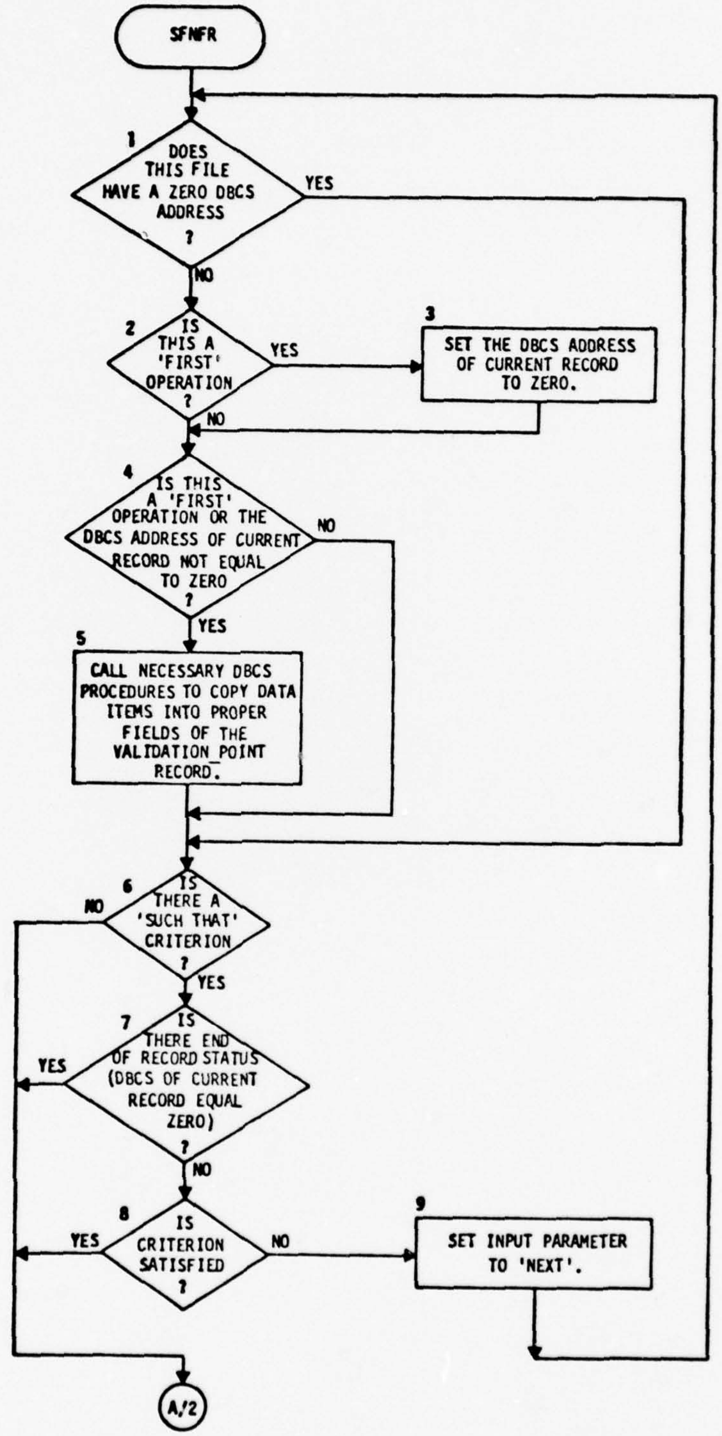



Figure 5-8 Select First/Next File Record (SFNFR)

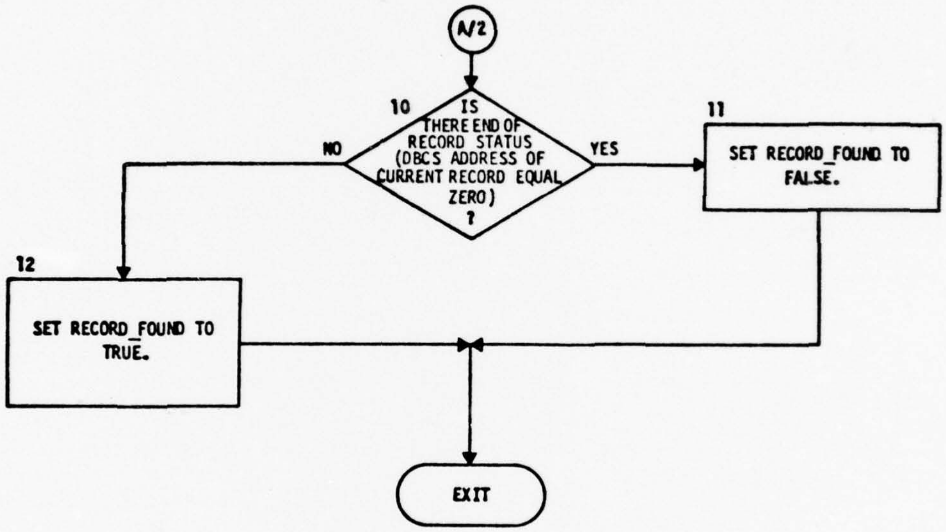



Figure 5-8 Select First/Next File Record (SFNFR) (Continued)

### 5.3.4 FOR EACH FILE RECORD (FEFR)

#### Description

The FOR EACH FILE RECORD module of the Data Manager is called during Simulator Post Processor execution by a user service request to gain access in a VALIDATION_POINT FILE to every instance (RECORD) or to every instance meeting some user specified criterion. For each RECORD which is found (or found to meet the criterion) a section of user supplied code is executed. The user supplied code may include other Data Manager service requests such as SELECT or FOR EACH. The FOR EACH criterion may be any valid Boolean expression but usually would contain one or more DATA items RECORDed in the FILE at a VALIDATION_POINT. All DATA items RECORDed in a VALIDATION_POINT FILE must be referenced as "VALIDATION_POINT name. DATA item name".

#### Input

Inputs to the FOR EACH FILE RECORD module are listed below:

- An optional criterion (called a 'SUCH THAT' clause) used to qualify which RECORDs are to be selected. Any Boolean expression is acceptable.
- A section of user PDL 2 source statements which may contain Data Manager service requests.

#### Processing

Processing performed in the FOR EACH FILE RECORD module is shown in the flow diagram of Figure 5-9. The following comments refer to processing box numbers.

| | | |
|---|---|---|
| [1],[6] | - | See the SELECT FIRST/NEXT FILE RECORD module (Section 5.3.3) for a detailed description. |
| [2] | - | Test is made on unqualified DATA item, RECORD_FOUND. |

#### Procedure Reference

In-line code except for calls to the SELECT FIRST/NEXT FILE RECORD module which are of the form VVYYYYY (YYYYY is the ordinal of the VALIDATION_ POINT FILE).

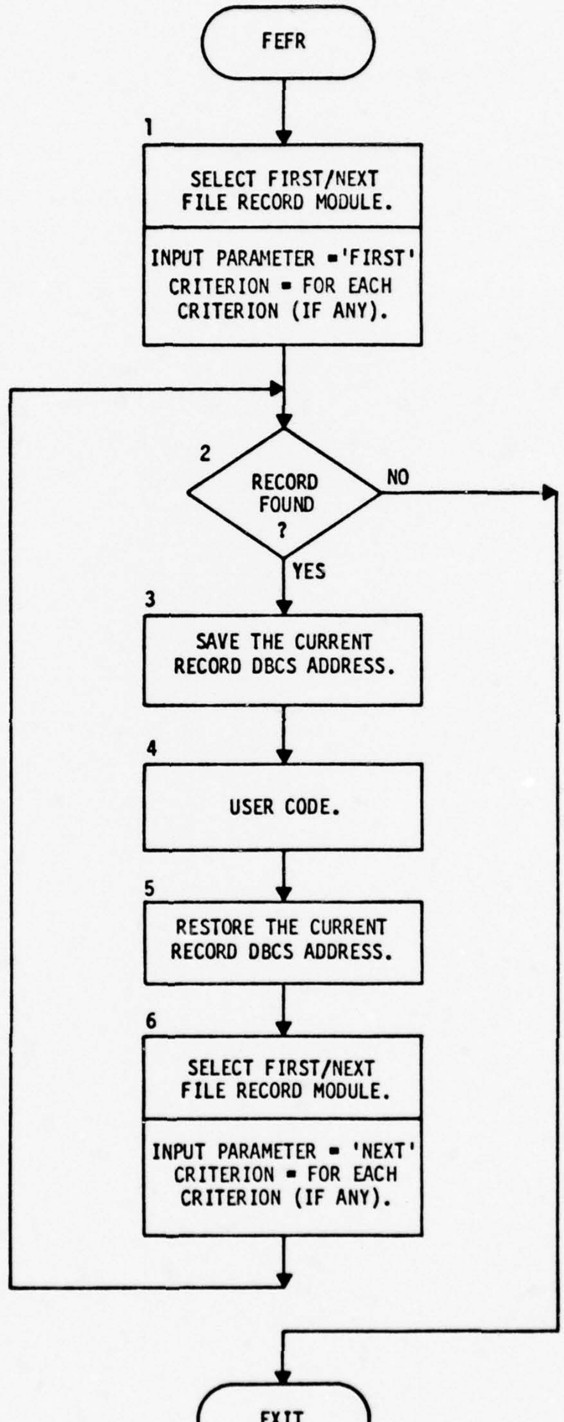



Figure 5-9 For Each File Record (FEFR)

## 6.0 INSTALLATION PROCEDURES

This section describes how to install and execute REVS on the ASC from the Software Deliverables File (SDF) delivered to the ARC, how to compile and reconstruct a new REVS load module, how to understand the macros used to invoke REVS, and what files are needed for execution. Comparable information is provided in Appendix B for installation of REVS on the CDC-7600 at the ARC.

### 6.1 SYSTEM INSTALLATION

REVS is designed to operate on the ASC, which requires that all files needed by a job be disk resident or staged from tape to disk prior to use. REVS has been delivered to the ARC on a Software Deliverables File tape (SDF) in accordance with BMDATC Software Standards [18] and is executable directly from this tape with the REVS macros only on disk or on cards. The installation procedures are therefore very simple in that no catalogued disk files are required at all.

In normal practice, the macros would likely be catalogued on disk as a minimum in order to avoid reproducing the cards in each run deck. The installer of REVS would add the REVS macros to the ASC macro file to make them available to all users. These macros are stored in source form in the REVS library REVSLIB which is the first file on the SDF tape. The macros are each flagged with the PDS 2 flag MACRO. Once the macros are included in the system macro library, REVS is fully usable from the SDF tape.

As documented in the REVS Users Manual [3], the REVS tape is specified on the REVSPREP macro with the tagged parameter REVSEFID. Therefore the following setup would be sufficient to run REVS from the SDF tape.

```
// JOB SAMPLE REVS SETUP FROM TAPE
// LIMIT BAND=150
// REVSPREP REVSEFID=SDF
```

In order to obtain the REVS macros from the SDF tape, the job displayed in Figure 6-1 may be used to list and punch all REVS macros.

### 6.2 PROGRAM CONSTRUCTION

REVS is a program described as a process in the PDS 2 [2] nomenclature. The library which contains REVS source is described in Section 7.1.

```
//   JOB   REVS MACROS RETRIEVAL JOB
//   LIMIT   BAND=200.MIN=5
//   MACASG   4.USERCAT/TRW/REVS/MACROS
//   COM
//   COM  LOAD NECESSARY FILE FROM SDF TAPE
//   COM
//   REVSPREP   REVSLIB=YES.REVSFID=SDF
//   COM
//   COM  FETCH MACRO  SOURCE FROM REVSLIB USING SLMS
//   COM
//   SLMS
*FETCH          REVSLIB(REVPREP):
*FETCH          REVSLIB(REVSXOT):
*FETCH          REVSLIB(SIMBUILD):
*FETCH          REVSLIB(SIMLOAD):
*FETCH          REVSLIB(SIMRUN):
*FETCH          REVSLIB(SIMSAVE);
*FETCH          REVSLIB (ASGASSM);
*FETCH          REVSLIB (TESTRUN);
//   COM
//   COM  LIST AND PUNCH MACROS
//   COM
//   RENAME   FT08F001.MACROS
//   EOSYS   MACROS.OPT=(K)
//   EOSYS   MACROS.TYPE=PUNCH
//   COM
//   COM
//   EOJ
```

Figure 6-1  Sample Job to Punch REVS Macros

The main program is named REVS, as can be seen by listing the configuration index. The job setup shown in Figure 6-2 is sufficient to compile and link edit a new version of REVS. The example will generate a load module with each function as an overlay and static memory buffers to permit up to 8 data base pages in memory. Changes to the Executive message XX 000 which defines the version number must be made in module XXUINIT. The REVS module has a NOLIST statement to suppress this module listing when partial compilations are performed. REVS is organized so that all functions (including the Executive) are at level 2 or lower in the process so that only the modules being changed require recompilation. Recompiling at the REVS level generates over 500 pages of compiler output. The PDS 2 feature of partial compilations is therefore of important value in such a large process. However, since the REVS module contains so many global declarations, the NOLIST statement suppresses the 700 lines of static declarations which otherwise result. The REVS module can be listed separately or the NOLIST can be changed to a LIST by a simple edit statement as is done in the example job shown in Figure 6-2.

## 6.3 JSL MACROS

There are 8 Job Specification Language (JSL) [4] macros in REVS which practically eliminate the need for user defined JSL when using REVS. These macros generate all necessary JSL statements needed to use REVS, including file acquisition, file saving, program execution, simulator building, and simulator execution. Only five of these macros are used directly by the REVS user (REVSPREP, REVSXQT, SIMRUN, TESTRUN, and ASGASSM) while the other three (SIMBUILD, SIMLOAD, and SIMSAVE) are lower level macros used by the three user macros. The user macros, including their parameters, are documented in the REVS Users Manual [3]. Additionally, each macro and its logic is more fully defined in this section.

### Macro REVSPREP

This macro acquires all files needed to run REVS and/or a REVS simulator and is used only once at the beginning of a job deck. It assigns and prints the default parameters, assigns system disk resident libraries, and then determines if REVS is running from tape, or the TRW development files on disk, as specified by the presence or absence of the REVSEFID parameter. If on tape, all necessary file definitions are generated and

```
//   JOB  REVS COMPILE AND LINKEDIT TO CONSTRUCT NEW LOAD MODULE
//   LIMIT   BAND=400.MIN=30
//   MACASG   M.USERCAT/TRW/REVS/MACROS
//   COM
//   COM    LOAD NECESARRY FILES FROM SDF TAPE
//   COM
//   REVSPREP   REVSLIB=YES.REVSEFID=SDF
//   COM
//   COM  COMPILE AND LINKEDIT REVS USING PDS2
//   COM
//   PDS2   CONFHEAP=20.CONFHSAP=20.:
     COMPHEAP=30.COMPTIME=700.COMPBAND=50/80/5.:
     OBJPHEAP=5,OBJPFSAP=15,OBJPRT=NO.OBJPTIME=120.:
     LNKTIME=500.LNKOPT=(M.A.Y,S,F,L).LSPACE=60
* LIBRARY     REVSLIB:
* USE         PROCESS      REVSYS:
* MODIFY      REVS:
* EDIT        "*NOLIST"/*:
* CATALOG     REVS:
* COMPILE     REVS(ALL):
* SWITCH      XXLNKCMD.    LIBRARY=REVSLIB:
//   COM
//   COM  SAVING REVS ON TAPE REQUIRES ALL 13 ASSOCIATED FILES
//   COM
//   MFR  REVSTAPE.L/RL=1/FLP/NEW
//   FOT  REVSLIB
//   FOT  SYS.LMOD
//   FOT  RSLDICT
//   FOT  DONNEES
//   FOT  RISF
//   FOT  LNKSIM
//   FOT  FT02F001
//   FOT  FT03F001
//   FOT  DBCS
//   FOT  JOBLIB
//   FOT  VVLIBE
//   FOT  NULLVVDB
//   FOT  FT11F001
//   MFRE
//   COM
//   COM
//   EOJ
```

Figure 6-2 Sample Job to Reconstruct REVS Load Module

then the REVS files are staged onto disk from the specified tape. If the files are the default TRW development disk files, these are assigned to the job and copies of REVSLIB and the ASSM are made. The REVSLIB parameter specifies whether or not the software development file is needed, which is the REVS source/object library REVSLIB. If this parameter is set to YES (by default or specification) these files are included. These files are only needed when making use of the REVS source library and are not acquired otherwise since they use a vast amount of disk storage.

The default ASSM is staged from the SDF tape but if a user selected ASSM is specified, another stage is generated to retrieve the specified ASSM. In addition (or instead, as the case may be), a load of a REVS simulator can be specified in which case the SIMLOAD macro is called to generate the JSL to stage the simulator from tape. Finally, the standard ARC libraries for graphics plotting and the PDS 2 macros are assigned.

It may be noted that the macro is really REVPREP with a synonym of REVSPREP. The former is used by the REVS software developers because it has defaults more appropriate for their use. The self documenting features of the macro clearly show the defaults for either macro name. The REVSPREP synonym only is documented in the Users Manual and the parameters that are relevant to the software developer are omitted from the macro to avoid confusion with production users. However, although the macro is programmed to execute differently for each name, it is the same macro and all parameters are effective with either macro even if they are not apparent in the default documentation (REVPREP reveals all parameters and default values).

Macro REVSXQT

This macro is used any number of times in a job to execute REVS. The REVS program must have been acquired by the REVSPREP macro and will have an access name of REVSABS. The macro generates the file definition statements needed during the REVS execution and the XQT statement. The PASCAL files INPUT and OUTPUT are nullified by the REVS Executive and are replaced with files REVSIN and REVSOUT in order to allow dynamic simulation building following SIMGEN concurrently with REVS execution. Since simulation building involves use of the PDL 2 compiler which is also a PASCAL program using the

same file names, REVS must not use these files simultaneously. After execution the macro generates the JSL statements for file disposition including printing of the REVS output files.

The macro is programmed to sense execution time conditions from the REVS Executive. If the Executive sets the JSL variable C to a non-zero value, then the macro will cause a Calcomp plot tape to be saved if given the option by the macro parameter CALCOMP (a non-zero value means at least one R_NET was plotted).

Likewise, if JSL variable B is set non-zero, it means SIMGEN has completed building a simulator and validator and they are ready to be compiled and link edited. Following REVS execution, the ASSM is saved if requested on the macro. Then macro SIMBUILD is called if a simulator has been built, and macro SIMSAVE called to save it on tape if specified by parameter SIMSAVE.

In addition to the default parameters available, all parameters defined for the PDL 2 compiler and linkage editor can be specified on the REVSXQT macro if necessary for the generated simulator.

Macro SIMRUN

This macro is used to execute a REVS simulator built by SIMGEN in a REVS execution. It calls the standard PDS 2 macro PXQT but provides self documenting defaults that are appropriate for a simulator. If recording data is generated, the validator data base is initialized.

Macro SIMBUILD

This macro generates the JSL to construct an absolute simulator and validator from the PASCAL source generated by SIMGEN. This macro is only used internally by the REVSXQT macro. SIMBUILD takes the simulator source file and passes it through the next utility of PDS 2 (unless suppressed by the parameter NEST) using the PXQT macro, then calls the PDL 2 compiler to compile the result. Following compilation, the PDL 2 object library is assigned and the linkage editor called using the ASC macro LNK. The necessary linkage editor directives are provided by the REVSPREP macro. After linkage editing, the access name of the simulator is SIMULATR unless modified by the parameter SIMNAME. Both the simulator source and object files are discarded by this macro in order to maintain the integrity of the simulator

and are not available for user manipulation. This macro is called a second time to produce the validator (simulator post processor) using linkage editor directives supplied by the REVS Executive.

Macro SIMLOAD

This macro generates JSL to load a simulator from tape and requires that REVSPREP precede it. It is in fact called by REVSPREP when SIMLOAD is YES and therefore not a user macro. A simulator always has an associated file (EEDF) that is unique to it and this is staged from tape with the simulator, along with the ASSM, validator, and validator data base.

Macro SIMSAVE

This macro generates the JSL to save a simulator on tape after creation by the SIMGEN function. In addition to the simulator and its associated file EEDF, the validator and validator data base are saved. SIMSAVE appends the three additional files on the tape which were used to generate the simulator: the ASSM from the simulator was generated (FT02F001), the Requirements Independent Source File (RISF), and the SETS Definition File (SDF). These files are provided at this time in order to support future configuration control procedures. This macro is internally called by the REVSXQT macro or SIMRUN macro if the parameter SIMSAVE is YES.

Macro TESTRUN

This macro is used to execute a validator program (simulator post processor) built by SIMGEN in a REVS execution. It calls the standard PDS 2 macro PXQT but provides self-documenting defaults that are appropriate for a validator.

Macro ASGASSM

This macro is provided to allow convenient acquisition of more than one ASSM in a job. This is not normally done in a batch environment, but is common when using REVS on the ASC at NRL. It assigns an ASSM from disk or tape depending on the parameters used, and creates a null ASSM if none are provided (with a warning note).

## 6.4 FILES

REVS requires a variety of files to be available depending on which functions are selected by a user. In order to simplify the file allocation,

and to avoid runtime errors in case of mis-specification, the REVSPREP macro acquires all files that might be used by REVS. Since these files are not large this does not cause any significant resource contention. The files that may be required on a run are files 2 through 13 of the Software Deliverables File (SDF) and are described in Section 7.1. These files are always acquired by the REVSPREP macro for REVS use. File 1 is only required to recreate REVS and is acquired only if REVSLIB = YES is specified on the REVSPREP macro. New versions of REVS must therefore conform to this file organization and save all 13 files as previously described in Section 6.2.

## 7.0 DETAILED DATA

Previous sections of this document describe the functional organization and operation of REVS and the installation of REVS from the Software Deliverables File. This section provides additional detailed information necessary for a maintenance activity on the ASC. Described are the organization of the REVS Software Deliverables File, the external files required for REVS execution and the use of the Lecarme-Bochmann Compiler Writing System and the ISDOS Data Base Control System. Comparable information is provided in Appendix B for REVS on the CDC-7600 Computer System.

### 7.1 SOFTWARE DELIVERABLES FILE

The tape constructed for delivery of the REVS software contains not only the source and data files in the format as required by the BMDATC standards [18], but also all relocatable and absolute modules corresponding to the source in order to facilitate the orderly use of the delivered system. The format of the tape is described in Figure 7-1. The content of each file of the tape follows (* denotes required files):

| | | | |
|---|---|---|---|
| 1 | * | REVSLIB | This is the PDS 2 source and object library which contains REVS. The modules are flagged according to the 17 flags shown in Figure 7-2. The PDS 2 library name is REVSLIB. The REVS process name is REVSYS. There are over 900 module names in this library which includes besides the PDS 2 source for REVS, the REVS macros, the SIMGEN card image files, and the ASSM data definition language schema. |
| 2 | | REVSABS | This is the executable load module of REVS and contains all REVS functions, using inter- and intra-function overlays. |
| 3 | * | RSLDICT | This is the dictionary file required by the RSL Translator function. It is a text file generated by the Lecarme-Bochmann Compiler Writing System (CWS). |
| 4 | * | DONNEES | This is the productions file required by the RSL Translator function. It is a PASCAL structured file generated by the CWS. |
| 5 | * | RISF | This is the requirements independent source file required by the Simulator Generation function. It is included in REVSLIB as source module GGRISF. |
| 6 | | LNKSIM | This is the Linkage Editor input file used to construct simulators. It is included in REVSLIB as source module LNKSIM. |

| File No. | Name | Type | BKSZ | LREC | RCFM | FORG | BAND (SECT) |
|---|---|---|---|---|---|---|---|
| 1 | REVSLIB | PDS 2 Library | 16384 | 256 | FB | DS | 100/200/5 |
| 2 | REVSABS | Load Module | 3840 | 80 | FB | DS | 12/60/2 |
| 3 | RSLDICT | Binary | 2000 | 80 | FB | PS | 1/5/1 |
| 4 | DONNEES | Binary | 1980 | 36 | FB | PS | 1/10/1 |
| 5 | RISF | Card Image | 4000 | 80 | FB | PS | 64/6400/64 (SECT) |
| 6 | LNKSIM | Card Image | 4000 | 80 | FB | PS | 64/6400/64 (SECT) |
| 7 | DB | Binary | 16384 | 4096 | FB | DS | 2/100/1 |
| 8 | DBT | Binary | 3840 | 10000 | VBS | PS | 64/6400/64 (SECT) |
| 9 | DBCS | Cifer Library | 3840 | 80 | FB | DS | 1/10/1 |
| 10 | JOBLIB | PDSAM Library | 3840 | 80 | FB | DS | 5/20/1 |
| 11 | VVLIBE | Cifer Library | 3840 | 80 | FB | DS | 1/10/1 |
| 12 | VVDB | Binary | 16384 | 4096 | FB | DS | 1/10/1 |
| 13 | VVDBT | Binary | 3840 | 10000 | VBS | PS | 64/6400/64 (SECT) |
| 14 | SPL | SMS-SPL | 3840 | 256 | FB | PD | 2/25/1 |
| 15 | CWS | SMS-SPL | 3840 | 256 | FB | PD | 50/100/5 |
| 16 | NUCLEUS | Card Image | 3840 | 80 | FB | PS | 1/5/1 |

Figure 7-1 Format of REVS Software Deliverables File (9 Track, 1600 BPI, NL)

FLAG ASSIGNMENT

| | | |
|---|---|---|
| 01 | PERM | PERMANENT MODULE |
| 02 | PDL | PDL MODULE |
| 03 | COMMON | COMMON MODULE |
| 04 | FORTRAN | FORTRAN MODULE |
| 05 | ALC | ASSEMBLER MODULE |
| 06 | PROCESS | PROCESS CONFIGURATION |
| 07 | REVSCON | REVS CONSTRUCT OPTIONS |
| 08 | RSL | RSL TRANSLATOR |
| 09 | ASSMDMP | ASSM DUMPER |
| 10 | TESTER | ASSM ACCESS TEST DRIVER |
| 11 | RNETGEN | *INTERACTIVE RNET GENERATOR* |
| 12 | RADX | REQ'MENTS ANALYSIS & DATA EXTRACTOR |
| 13 | SIMGEN | SIMULATION GENERATION |
| 14 | SIMXQT | SIMULATION EXECUTION |
| 15 | MACRO | REVS ASC JSL MACROS |
| 16 | DDL | ASSM DATA DEFINITION LANGUAGE (DDL SCHEMA) |
| 17 | CALCOMP | CALCOMP FUNCTION FOR PLOTTING STRUCTURES |
| 18 | SIMDA | SIMULATION DATA ANALYSIS |
| 19 | NRLPLOT | NRL PLOT INTERFACE |
| 20 | UPDSMS | CHARACTER SET/LIBRARY FORMAT CONVERSION UTILITY |
| 21 | VALIDATR | VALIDATION POST PROCESSORS |

Figure 7-2 REVSLIB Module Flags

| | | |
|---|---|---|
| 7 | * DB | This is the direct access ASSM file which is named FT02F001 during execution of REVS. It contains the RSL nucleus definition only. It is created by the use of the DBCS utilities in JOBLIB, and the REVS function RSLXTND. |
| 8 | * DBT | This is the binary data base table file required by the ASSM access procedures of REVS and is named FT03F001 during execution. |
| 9 | DBCS | This is the object module library for the Data Base Control System (DBCS) which is required for REVS. |
| 10 | JOBLIB | This is the absolute program library for the DBCS and contains the utilities for initializing data bases for REVS use, and programs to support SIMGEN and post processing. |
| 11 | VVLIBE | This is the object module library for the validators (simulation post processors) generated by REVS. |
| 12 | * VVDB | This is the direct access recording file which is named FT10F001 during execution of REVS. It contains the recordings obtained from a simulator execution. It is initially built by the DBCS utilities in JOBLIB. |
| 13 | * VVDBT | This is the binary data base table file required by the Validator data base builder in JOBLIB and any validator program. |
| 14 | * SPL | This is the FORTRAN source library for the DBCS used by REVS. This library must be compiled by the ASC NX compiler as the FX compiler generates incorrect code. There are 176 modules in this library. |
| 15 | * CWS | This is the compiler writing system used to generate the REVS RSL Translator function and its two required files. It is all in PASCAL and contains 11 modules. |
| 16 | NUCLEUS | This is the card image file which defines RSL. It was used as input to RSLXTND to create the initial ASSM (DB). |

## 7.2 REVS EXTERNAL FILES

REVS is designed to maintain a software requirements data base, the Abstract System Semantic Model (ASSM). In addition to the ASSM, five external files are input to REVS. This section describes the structure of the ASSM and the contents and source of the additional input files. In addition

to print files and the ASSM, REVS can generate two additional output files: a Simulator Program file (See Sections 3.5 and 6.3) and a CALCOMP file (See Section 3.3).

### 7.2.1 Abstract System Semantic Model (ASSM)

The ASSM is a data base system designed to support the Requirements Statement Language (RSL) which allows the requirements engineer to state requirements specifications over a period of time as they are developed and to do so in a non-procedural manner. It provides a central repository wherein requirements are collected and maintained in an abstract, relational model. The RSL statements that the requirements engineer inputs to REVS are analyzed and a representation of the semantics are placed in the ASSM.

The ASSM is specifically designed to be initialized and controlled by the ISDOS Data Base Control System (DBCS) (See Section 7.3.2). The structural organization of the ASSM is shown in Figure 7-3. Logical records, which are the basic building blocks of the ASSM, are represented as annotated ovals. In DBCS terminology, records may be members of a group of records owned by some other record. Such a group of member records, along with its owner record, is referred to as a set. A set is illustrated in Figure 7-3 via the annotated directed arcs connecting various ASSM records. The owner record in the set is at the tail of the arc while member records are at its head. The ASSM set is given by the annotation along the arc. Note that records may be owners in more than one set and/or members of more than one set. However, any given record can be a member of only one instance of any given set. ASSM records and sets are organized to reflect and support the RSL primitives (elements, relationships, attributes, and structures).

Following is a description of each ASSM record identified in the ASSM structure (Figure 7-3).

SYSTEM - This is a unique record in the ASSM which is automatically generated by the DBCS at data base initialization time. There is only one instance of this record in the data base and it may be an owner for any number of sets, but can never be a member record (owned record) of any set. It contains no data other than the appropriate record pointers generated by DBCS for each of the sets for which it is an owner.

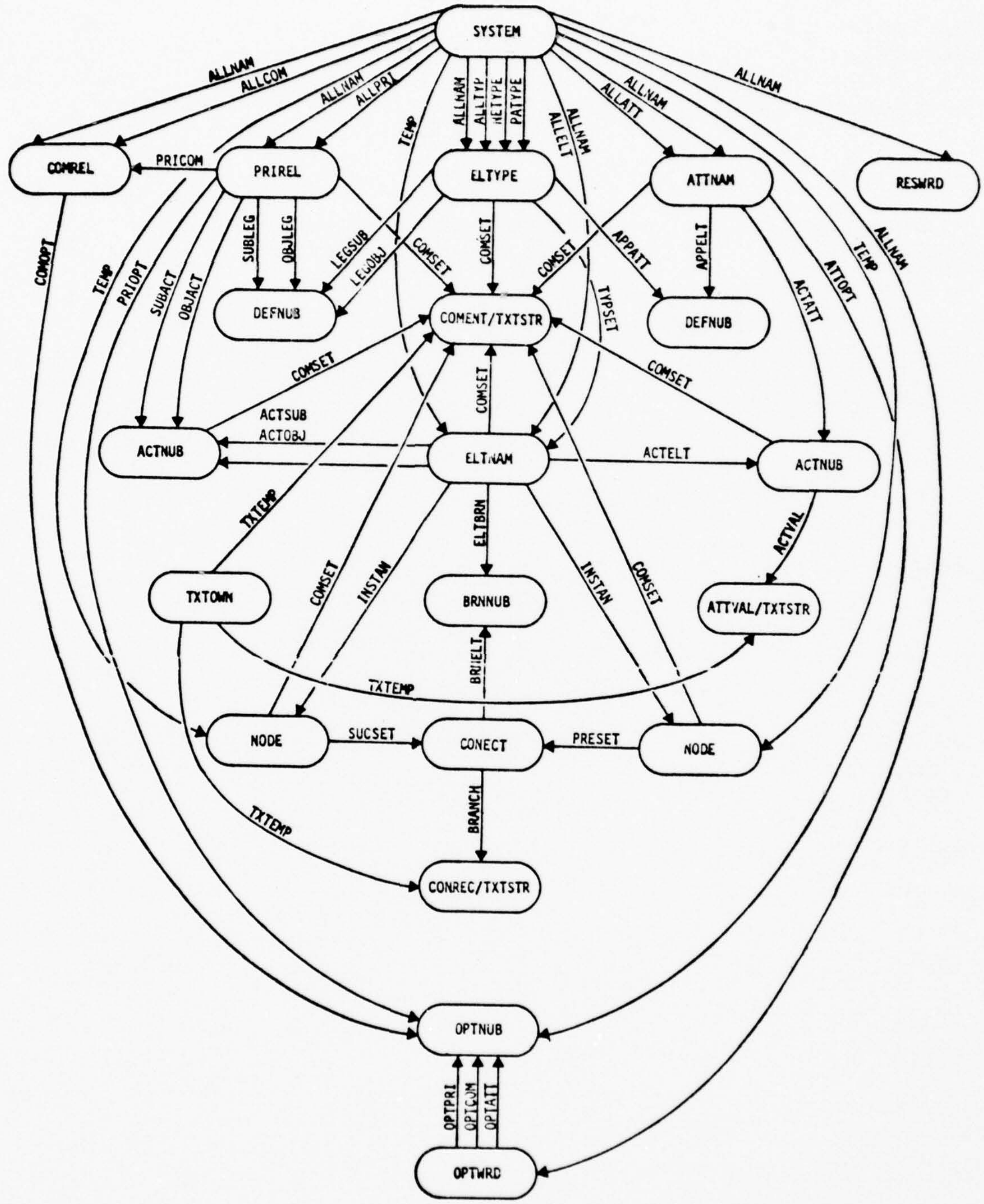



Figure 7-3 ASSM Configuration

ELTYPE - For each RSL element type defined by the user, an ELTYPE record containing the element type name is created in the ASSM.

ATTNAM - For each RSL attribute defined by the user, an ATTNAM record containing the attribute name is created in the ASSM.

PRIREL - For each RSL relationship defined by the user, a PRIREL record containing the primary relationship name is created in the ASSM.

COMREL - For each RSL relationship in the ASSM, the user may define a complementary relationship which will result in the creation of a COMREL record containing the complementary relationship name.

ELTNAM - For each RSL element defined by the user, an ELTNAM record containing the element name is created in the ASSM.

OPTWRD - For each RSL optional word defined via an attribute definition or relationship definition, an OPTWRD record containing the optional word is created in the ASSM.

RESWRD - For each reserved word defined in the Requirements Statement Language, a RESWRD record containing the applicable reserved keyword will be created in the ASSM.

NODE - For each node appearing in the structure declaration of an R_NET, SUBNET, or VALIDATION_PATH element, a node record is created in the ASSM. The node record will provide for at least the following data items - node type, node color, and x, y screen position, if applicable.

CONECT - This record is created to provide for successor/predecessor relationships between nodes in a structure. The record will provide for the x, y screen positions of the end points of the directed arc when displayed on the Anagraph.

CONREC - This record, or group of records, will contain the conditional expression for the branch of an OR/FOR node in an R_NET/SUBNET structure.

COMENT - This record, or group of records, will contain comments consisting of textual data supplied by the user at any level of RSL definition.

BRNNUB
DEFNUB
ACTNUB
OPTNUB - These are artificial records required by the DBCS for defining and maintaining m by n relationships in the ASSM. Their contents consist of record pointer information generated by DBCS as set instances are created in the ASSM.

ATTVAL - As attribute instances are defined by the user, this record, or group of records (each record contains an actual attribute value), is created in the ASSM.

TXTOWN - As arbitrary text strings are entered by the user, this record is created temporarily as the owner of the text string. The record is subsequently discarded when the text string is attached to some existing record in the ASSM.

TXTSTR - This record, or group of records, offers an additional technique of entering textual data in the ASSM. The record(s) will contain textual data such as comments or attributes with value of TEXT.

STAT - This record is created upon the first call to each of the ASSM access procedures. Its contents, usage count, is updated on each subsequent call to an ASSM access procedure.

Following is a description of each of the sets identified in the ASSM structure (Figure 7-3).

ALLNAM - This set consists of an alphabetical list of all currently defined RSL words. Its owner is the DBCS SYSTEM record and its member records are element type names (ELTYPE), attribute names (ATTNAM), relationship names (PRIREL, COMREL), element names (ELTNAM), optional words (OPTWRD), and reserved keywords (RESWRD).

ALLPRI - The ALLPRI set consists of an alphabetical list of all currently defined RSL relationships in the ASSM. It is owned by the DBCS SYSTEM record with members consisting of all primary relationship names (PRIREL records).

ALLCOM - The ALLCOM set consists of an alphabetical list of all currently defined complementary relationships in the ASSM. It is owned by the DBCS SYSTEM record with members consisting of all complementary relationship names (COMREL records).

SUBLEG
OBJLEG
LEGSUB
LEGOBJ - These sets are used to define the legal subject element types and legal object element types for a particular relationship in the ASSM.

| | |
|---|---|
| SUBACT<br>OBJACT<br>ACTSUB<br>ACTOBJ | These sets are used to define an instance of a relationship in use in the ASSM. |
| PRICOM - | This set is used to correlate a primary relationship with its corresponding complementary relationship. |
| OPTPRI<br>PRIOPT<br>OPTCOM<br>COMOPT | These sets are used to relate optional words with primary and complimentary relationships as defined by the user. |
| ALLATT - | This set consists of an alphabetical list of all currently defined RSL attributes in the ASSM. It is owned by the DBCS SYSTEM record with members consisting of all attribute names (ATTNAM records). |
| APPELT<br>APPATT<br>LEGVAL | These sets are used to define the legal applicable element types and legal values for a particular attribute name in the ASSM. |
| ACTATT<br>ACTELT<br>ACTVAL | These sets are used to define an instance of an attribute for a given element in the ASSM. |
| ATTOPT<br>OPTATT | These sets are used to associate optional words with an attribute name as defined by the user. |
| COMSET - | This set provides for comments to be associated with RSL elements, element types, attribute definitions, relationship definitions, structure nodes, attribute instances, and relationship instances. |
| ALLELT - | The ALLELT set consists of an alphabetical list of all currently defined RSL elements in the ASSM. It is owned by the DBCS SYSTEM record with member records consisting of all element names (ELTNAM records). |
| TYPSET - | This set defines the element type for each RSL element as specified by the user via RSL. |
| ALLTYP - | The ALLTYP set consists of an alphabetical list of all currently defined RSL element types in the ASSM. It is owned by the DBCS SYSTEM record with member records consisting of all element type names (ELTYPE records). |
| INSTAN - | When applicable, this set specifies the ASSM element referenced at a node on the structure declaration of an R_NET, SUBNET, or VALIDATION_PATH. The set is owned by the ASSM element name record (ELTNAM) and has as its members all node records (NODE) for any and all structures in the ASSM which reference the ASSM element. |

SUCSET }
PRESET } These sets provide the successor/predecessor relationship between nodes on a structure declaration.

BRANCH - This set is used to associate a conditional expression with each branch of an OR node. The set is owned by the connector record (CONECT) and has as its member a conditional expression record or group of records (the CONREC record).

TEMP - This set is used for constructing structures. Once the structure is complete and has been saved, the TEMP set is deleted.

STATS - This set contains all the STAT records which contain a tally of the usage of each of the ASSM access procedures.

NETYPE - This set specifies an alphabetical list of all the element types which are allowed on the structure of an R_NET or SUBNET.

PATYPE - This set specifies an alphabetical list of all the element types which are allowed on the structure of a VALIDATION_PATH.

TXTEMP - This set is used for temporarily constructing a text string.

ELTBRN }
BRNELT } These sets are used to provide the association between RSL elements and their reference by conditional expressions on an OR/FOR branch within a structure.

### 7.2.2 RSL Translator Input Files

The RSL Translator requires two standard PDL 2 input files, both of which are constructed during the generation of the translator (see Section 7.3.1) and read in during the initialization phase of each translator invocation. The file DONNEES is created by the syntactic analysis generation (SYNTGEN) and lexical analysis generation (LEXIGEN) phases of the Lecarme-Bochmann Compiler Writing System (L-B CWS). This file is a PDL 2 structured binary (non-text) file and contains the information necessary to initialize the lexical and syntactic analysis tables used by the translator.

The second required input file is the text file RSLDICT, which contains a dictionary of RSL keywords recognized by the lexical analyzer and the syntactic symbols used by the syntax analyzer. This file is constructed by the syntactic analysis generation (SYNTGEN) program of the L-B CWS.

RSLDICT is used by the translator to initialize a dictionary array. The array in turn is used to provide an intelligible display of the contents of the parse stack and window whenever a dump of the stack is requested (see Section 3.2).

### 7.2.3 Simulator Generation Input Files

The Simulator Generation function uses two PDL 2 text files as input. They are:

a) the Requirements Independent Source File (RISF), and

b) the SETS Definition File (SDF).

The RISF contains source code for components of the Simulator Program that are independent of the particular requirements model being generated. It is included in REVSLIB as source module GGRISF (see Section 7.1). The text of the RISF is separated into segments by the character "$". Each segment is inserted into its proper place in the Simulator Program during the consolidation phase of Simulator Generation. The SIMGEN Consolidation module uses the "$" characters to recognize the ends of segments. Figure 7-4 shows the format of the RISF and identifies the content of each segment.

The SDF contains the source code for the components of the Simulator Program that represent SETS. This file is constructed externally to REVS and, like the RISF, is separated into segments by the character "$". Each segment of the SDF is inserted into its proper place in the Simulator Program during the consolidation phase of Simulator Generation. Figure 7-5 shows the format of the SDF and identifies the content of each segment. Note that the PDL 2 keywords LABEL, CONST, TYPE, and VAR which are supplied as part of the RISF (see Figure 7-4) are not repeated for the segments composing the SDF.

## 7.3 SUPPORT SOFTWARE/UTILITIES

Two generally available software packages, the Lecarme-Bochmann Compiler Writing System and the Data Base Control System, are used to support the operation and maintenance of REVS. This section details the manner in which these packages are employed.

```
PROGRAM EEPROGRAM (OUTPUT);
LABEL
LABEL DECLARATIONS
CONST
CONSTANT DECLARATIONS
$
TYPE
TYPE DECLARATIONS
$
VAR
VARIABLE DECLARATIONS
$
DATA MANAGEMENT PROCEDURES
$
DATA RECORDING PROCEDURES
$
EVENT MANAGEMENT PROCEDURES
$
SIMULATOR INITIALIZATION PROCEDURES
$
SIMULATOR EXECUTIVE UTILITY PROCEDURES
$
PROCEDURE SCHEDULER BEGINNING CODE
$
PROCEDURE SCHEDULER END CODE
$
SIMULATOR EXECUTIVE PROCEDURE
$
SIMULATOR PROGRAM BODY
$
```

Figure 7-4 Format of RISF

```
CONSTANT DECLARATIONS
$
TYPE DECLARATIONS
$
VARIABLE DECLARATIONS
$
INITIALIZATION PROCEDURES
$
MODEL PROCEDURES
$
```

Figure 7-5 Format of SDF

### 7.3.1 Lecarme-Bochmann Compiler Writing System

The RSL Translation function of REVS has been constructed using the facilities provided by the Lecarme-Bochmann Compiler Writing System (L-B CWS) [16, 17]. The L-B CWS accepts an integrated description of the syntax and semantics of a language and produces a compiler or translator, written in PASCAL, for that language. The lexical analyzer, syntactic analyzer (parser), and error handling procedures are automatically supplied by the L-B CWS, freeing the language designer to concentrate on the area he is responsible for, the semantics or meaning of the language. The use of the L-B CWS has greatly simplified the construction of the RSL translator and provides for the relatively easy modification and evolution of RSL.

The use of the L-B CWS to construct the RSL translator is shown in Figure 7-6, which is a listing of a sample deck. The only required modification of this sample deck is the substitution of an appropriate JOB card and the substitution of the actual REVS Software Deliverables File tape number for the mnemonic designation of SDF on the fourth and eighth cards in the listing.

The following comments are keyed to the annotations on the figure. Note, however, that familiarity is assumed with JSL on the ASC [4] as well as the SMS [9] and PDS [2] utilities used by the run stream. Familiarity is also assumed with the general construction of the L-B CWS and its use [17, 19].

[1] - Acquire system resources. The necessary macros are assumed to exist on a disk file.

[2] - The REVPREP macro call causes standard REVS files to be brought in from the SDF and performs necessary initializations. The RSL translator input files DONNEES and RSLDICT are released as they will be reconstructed by this run.

[3] - Access the SMS SPL file containing the source of the L-B CWS and all its input files. Also access the NUCLEUS file which contains the pre-defined RSL element types, attributes, and relationships. This file will be used in the last step of the run to test the generated translator and construct a baseline ASSM.

[4] - Assign the standard PDL 2 library file.

[5] - Obtain the SEMAGEN program source, compile and link edit the program, name the load module SEMALMOD.

```
[1]  // JOB RIN-14-BUILD,PTLUJ,GUNTHER
     // LIMIT BAND=700,MIN=30
     // MACASG MACROS.USERCAT/TRW/REVS/MACROS
[2]  // REVPREP REVSEFID=SDF
     // REL DONNEES,RSLDICT
[3]  // FD SPL.FORG=PO.BKSZ=3840.LREC=256,RCFM=FB,BAND=50/75/5
     // FD NUCLEUS,BKSZ=3840.LREC=80.RCFM=FB.FORG=PS,BAND=1/5/1
     // MFR FILES.LABL=1/NL.EFID=SDF
     // FIT SPL.LABL=15/NL
     // FIT NUCLEUS,LABL=16/NL
     // MFRE
[4]  //  ASG      PLIB.  USERCAT/TIABMDA/PDS/PDL2/LIBRARY,USE=SHR
[5]  // SMS COMPILE=80
     $0 B.E.8
     $EDIT SEMAGEN
     // RENAME COMPILE.SEMAGEN
     // PDL2 INPUT=SEMAGEN
     // LNK LNKOPT=(M,A,Y,S.E.L)
      LIBRARY PLIB
      INCLUDE MAIN$$
     // REL SEMAGEN
     // RENAME SYS.LMOD,SEMALMOD
[6]  // SMS COMPILE=80
     $0 B.E.8
     $EDIT RSLSYN
     // RENAME COMPILE.RSLSYN
     // PXQT INPUT=RSLSYN.OPT=(Z),STKSIZE=8000,ADDMEM=20K,GO=SEMALMOD
     // REL RSLSYN
[7]  // SMS COMPILE=80
     $0 B.E.8
     $EDIT SYNTGEN
     // RENAME COMPILE.SYNTGEN
     // PDL2 INPUT=SYNTGEN.COMPSTK=40
     // LNK LNKOPT=(M.A,Y.S,E.L)
      LIBRARY PLIB
      INCLUDE MAIN$$
     // REL SYNTGEN
     // RENAME SYS.LMOD.SYNTLMOD
[8]  // PXQT OPT=(Z),STKSIZE=16000,ADDMEM=20K,OJTBAND=8/20/2,GO=SYNTLMOD
     // FD DONNEES.POS=MOD
[9]  // SMS COMPILE=80
     $0 B.E.8
     $EDIT LEXIGEN
     // RENAME COMPILE.LEXIGEN
     // PDL2 INPUT=LEXIGEN
     // LNK LNKOPT=(M,A,Y,S,E,L)
      LIBRARY PLIB
      INCLUDE MAIN$$
     // REL LEXIGEN
     // RENAME SYS.LMOD.LEXILMOD
[10] // SMS COMPILE=80
     $0 B.E.8
     $EDIT NOYALEX
     // RENAME COMPILE.NOYALEX
     // PXQT OPT=(Z),STKSIZE=12000,ADDMEM=20K,GO=LEXILMOD
[11] // SMS COMPILE=80
     $0 B.E.8
     $EDIT COMPGEN
     // RENAME COMPILE.COMPGEN
     // PDL2 INPUT=COMPGEN
     // LNK LNKOPT=(M.A,Y,S,E,L)
      LIBRARY PLIB
      INCLUDE MAIN$$
     // REL COMPGEN
     // RENAME SYS.LMOD,COMPLMOD
```

Figure 7-6 Sample Job for RSL Translator Construction

```
[12] // SMS COMPILE=80
     $O B,E,8
     $EDIT RSLGLO
     // RENAME COMPILE,GLOBAUX
     // SMS COMPILE=80
     $O B,E,8
     $EDIT NOYASYN
     // RENAME COMPILE,NOYASYN
     // SMS COMPILE=80
     $O B,E,8
     $EDIT ERREURS
     // RENAME COMPILE,ERREURS
[13] // FD COMPILE,BAND=5/30/5,LREC=80,BKSZ=2000,FORG=PS,RCFM=FB
[14] // PXOT OPT=(Z),STKSIZE=8000,ADDMEM=25K,HEAPSIZE=5000,GO=COMPLMOD
     // RENAME COMPILE,TTRSL
[15] // SMS COMPILE=80
     $O B,E,B
     $EDIT PROGGEN
     // RENAME COMPILE,PROGGEN
     // PDL2 INPUT=PROGGEN
     // LNK LNKOPT=(M,A,Y,S,E,L)
      LIBRARY PLIB
      INCLUDE MAIN$$
     // PEL PROGGEN
     // RENAME SYS.LMOD,PROGLMOD
[16] // ASG SPLITPDL,USERCAT/TIABMDA/PDS/PASCAL/SPLITPDL,USE=SHR
     // FD PUNCH,LREC=80,BKSZ=4000,RCFM=FB,BAND=1/15/1
     // PXOT GO=SPLITPDL,STKSIZE=10000,INPUT=TTRSL,CPTIME=200
[17] // FITLIB TTLIBE
     // CONFIG CONFHEAP=30,CONFBSAP=30,CONFTIME=250
     *BUILD PROCESS TTSYS;
     *INITIALIZE LIBRARY TTLIBE;
     *LIBRARY TTLIBE;
     *OPTIONS NOLISTM;
     *ADD SONS TTRSL;
     *SWITCH PUNCH;
     *CATALOG PROCESS TTSYS;
     *NEST TTRSL(ALL),DECLARATION=(4,80),BODY=(2,0);
[18] // PDS2  CONFHEAP=20,CONFBSAP=20,CONFTIME=200,;
     COMPHEAP=35,;
     OBJPPT=NO,OBJPHEAP=10,OBJPBSAP=20,OBJPTIME=120,;
     LNKPRT=NO,LNKTIME=700,LNKOPT=(M,A,Y,S,E,L),LSPACE=70,LNKCMD=OVLYCMDS
     *LIBRARY REVSLIB;
     *USE PROCESS REVSYS;
     *MERGE PROCESS TTSYS,LIBRARY=TTLIBE;
     *CATALOG PROCESS;
     *SWITCH XXLNKCMD,LIBRARY=REVSLIB;
[19] // FD DONNEES,POS=NEW
     // RENAME SYS.LMOD,REVSLMOD
[20] // FXOT GO=OBIN
     // REVSXOT GO=REVSLMOD,CPTIME=900
[21] RSLXTND.
     ADDFILE TRANSPARENT NUCLEUS.
     FEND.
     RADX.
     LIST RSL ALL.
     LIST ALL.
     FEND.
     STOP.
     // EOJ
     *
```

Figure 7-6 Sample Job for RSL Translator Construction (Continued)

[6] - Obtain the integrated description of RSL (RSLSYN), execute the SEMAGEN program using RSLSYN as the input file.

[7] - Obtain the SYNTGEN program source, compile and link edit the program, name the load module SYNTLMOD.

[8] - Execute the SYNTGEN program. The only input files are those constructed by SEMAGEN. Note that the output file DONNEES must be left positioned at the end (POS=MOD) so that the next program (LEXIGEN) can also write on it.

[9] - Obtain the LEXIGEN program source, compile and link edit the program, name the load module LEXILMOD.

[10] - Obtain the standard lexical nucleus file NOYALEX and execute the LEXIGEN program.

[11] - Obtain the COMPGEN program source, compile and link edit the program, name the load module COMPLMOD.

[12] - Obtain the file of RSL global semantic actions (called RSLGLO on the SPL, known as GLOBAUX by the L-B CWS). Also obtain the syntax nucleus file NOYASYN and the standard error nucleus file ERREURS.

[13] - The program COMPGEN constructs the generated translator on the file named COMPILE. The FD insures that sufficient space is allocated for the file.

[14] - Execute the COMPGEN program, rename the generated translator TTRSL.

[15] - Obtain the PROGGEN program source, compile and link edit the program, name the load module PROGLMOD. Note that this program is not executed in this run; this sequence is only included to indicate how the load module may be constructed. The PROGGEN program is useful in checking out proposed syntax changes to the language and in building test inputs for the translator. In actual use the file RSLDEF, a deck on this SPL which contains a syntax description of RSL but no semantics, is used in place of RSLSYN as the input to the SEMAGEN program, which must be run, along with SYNTGEN and LEXIGEN, before PROGGEN can be executed. (See the L-B CWS User's Manual [17] for a discussion of the use of PROGGEN.)

[16] - Obtain and execute the SPLITPDL program on the translator file (TTRSL). This program converts the standard PASCAL program TTRSL into a PDS 2 compatible format.

[17] - Execute the PDS 2 Configuration Processor to construct a PDS 2 process named TTSYS and a PDS 2 library named TTLIBE.

[18] - Execute the PDS 2 system to merge TTSYS with the REVS process REVSYS. The commands on XXLNKCMD cause compilation and linkage editing of the REVS program.

[19] - Reposition the file DONNEES to the beginning (POS=NEW) so that it can be read by the RSL translator.

[20] - Execute the REVS utility program DBIN to construct an empty data base (ASSM). Then execute the constructed REVS load module starting from this data base.

[21] - The RSL translator is used to input the RSL NUCLEUS file, then the Requirements Analysis and Data Extraction Function (RADX) is used to list out the contents of the ASSM.

### 7.3.2 Data Base Control System

The Data Base Control System (DBCS) is the FORTRAN data base system used by REVS to maintain the ASSM, and is documented in ISDOS Working Paper number 88 from the University of Michigan [15]. The entire source for the DBCS is contained in the SMS program library on file 14 of the Software Deliverables File (SDF) (see Section 7.1). The NX compiled relocatable object library is on file 9 and the absolute utilities are in the JOBLIB on file 10. The JOBLIB and object library are automatically provided by use of the REVSPREP macro.

Two DBCS utilities are used to prepare an initial data base for use by REVS. The first utility, the Data Definition Language Analyzer (DDLA), translates the data base schema (DDL) into the data base dictionary, the Data Base Table file (DBT), for subsequent use by the DBCS. The second utility, the Data Base Initialization (DBIN) program, creates a null Data Base file (DB) based on the DBT created by DDLA. These two files are then used by REVS to build and maintain an ASSM. The DB file is FORTRAN unit 2 and the DBT file is FORTRAN unit 3. The DBIN program can be executed on an existing data base in which case it will still create a null DB file over the existing file. The DBT is a read only file for DBIN and REVS and is file number 8 on the SDF. The data base on file 7 of the SDF is an ASSM which contains only the RSL definition as documented in the REVS Users Manual. DBIN can be used to generate a completely null data base.

The deck setup shown in Figure 7-7 illustrates the process of building the nominal ASSM which is on file 7 of the SDF. The DDL is obtained from REVSLIB (module XXDDL), the utilities are in the JOBLIB, REVS is the absolute program and the RSL definition is on file 16 of the SDF.

```
//   JOB   DBCS UTILITIES TO INITIALIZE ASSM DATA BASES
//   LIMIT   BAND=200,MIN=15
//   MACASG   M.USERCAT/TRW/REVS/MACROS
//   COM
//   COM    LOAD NECESSARY FILES FROM SDF TAPE
//   COM
//   REVSPREP   REVSFFID=SDF
//   FD   RSLDEF,BKSZ=3840,LREC=80,RCFM=FB,BAND=1/5/1
//   FTIP   RSLDEF,LABL=16/NEXTFID=SDF
//   RENAME   FT02F001,DB
//   RENAME   FT03F001,DBT
//   FD   FT02F001,BAND=1/30/1,FORG=DS,BKSZ=15384,LREC=4096,RCFM=FB
//   FD   FT03F001,BAND=1/30/1,BKSZ=03840,LREC=10000,RCFM=VBS
//   COM
//   COM    EXTRACT DDL FROM REVSLIB TO DDL
//   COM
//   SLMS
*FETCH REVSLIB(XXXDDL);
//   RENAME   FT08F001,DDL
//   COM
//   COM    EXECUTE DDLA USING DDL: CREATE DBT ON FT03F001
//   COM
//   EXQT   GO=DDLA,DATA=DDL
//   COM
//   COM    EXECUTE DBIN USING DBT: CREATE NULL DB ON FT02F001
//   COM
//   EXQT   GO=DBIN
//   COM
//   COM    EXECUTE REVS USING RSLDEF: CREATE ASSM (NOT SAVED)
//   COM
//   WAIT
//   REVSXQT
RSLXTND.  ADD RSL DEFINITION TO NULL DB.
ADDFILE TRANSPARENT RSLDEF.  GET INPUT FROM SDF FILE 16
RADX.   LIST RESULTANT ASSM CONTENTS.
LIST RSL ALL.  LIST ALL.
STOP.  END OF ASSM INITIAL BUILD.
//   COM
//   COM    COULD HAVE SAVED ASSM IN REVSXQT WITH ASSMSAVE=YES
//   COM
//   COM    FT02F001 CORRESPONDS TO SDF FILE 7 (DB)
//   COM    FT03F001 CORRESPONDS TO SDF FILE 8 (DBT)
//   COM
//   COM
//   COM
//   COM
//   EOJ
```

Figure 7-7  Sample Job to Initialize ASSM

The deck setup shown in Figure 7-8 illustrates the process of building a null recording data base for the REVS post processor programs. The DDL is obtained from REVSLIB (module VVDDL), the utilities are in the JOBLIB. The resulting VV data base (FT10F001) and data base tables (FT11F001) reflect those which are on files 12 and 13, respectively, of the SDF.

```
// JOB      DBCS UTILITIES TO INITIALIZE VV DATA BASE
// LIMIT    BAND=200, MIN=15
// MACASG   M, USERCAT/TRW/REVS/MACROS
// COM
// COM      LOAD FILES FROM SDF TAPE
// COM
// REVSPREP REVSLIB=YES, REVSEFID=SDF
// REL      FT02F001
// REL      FT03F001
// FD       FT02F001, BAND=1/30/1, FORG=DS, BKSZ=16384, LREC=4096, RCFM=FB
// FD       FT03F001, BAND=1/30/1, BKSZ=3840, LREC=10000, RCFM=VBS
// COM
// COM      EXTRACT DDL FROM REVSLIB
// COM
// SLMS
*FETCH REVSLIB (VVDDL);
// RENAME   FT08F001, DDL
// COM
// COM      EXECUTE DDLA USING DDL; CREATE DBT ON FT03F001
// COM
// FXQT     GO=DDLA, DATA=DDL
// COM
// COM      INITIALIZE VV DATA BASE ON FT02F001 USING DBIN
// COM
// FXQT     GO=DBIN
// REL      FT11F001
// COM
// COM      THE DATA BASE AND DATA BASE TABLES MUST BE ON
// COM      FORTRAN UNITS FT10F001 and FT11F001, RESPECTIVELY
// COM
// RENAME   FT02F001, FT10F001
// RENAME   FT03F001, FT11F001
// EOJ
```

Figure 7-8  Sample Job to Initialize VV Data Base

# 8.0 CHANGE CONSIDERATIONS

REVS is designed to be independent of the definition of RSL where possible. The degree of independence varies for the different functions which compose REVS. Should any changes be made to the definition of RSL, an evaluation of the dependent REVS functions must be made to determine the necessary modification that should be reflected in the function. Table 8.1 summarizes in matrix form those functions which can be impacted by changes to the RSL components labeling the rows of the matrix. The remainder of this section identifies by REVS function the concepts and keywords of RSL that the function requires.

## 8.1 REVS EXECUTIVE CHANGE CONSIDERATIONS

The REVS Executive is not dependent on any of the keywords or concepts which define RSL.

## 8.2 RSL TRANSLATOR CHANGE CONSIDERATIONS

The following RSL names receive special handling by the RSL Translator. If any of them are changed in extension mode, the translator code must be modified to reflect the change.

| | |
|---|---|
| Value names: | BOOLEAN<br>ENUMERATION<br>NAMED<br>NUMERIC<br>TEXT |
| Element-type-names: | ALPHA<br>DATA<br>ENTITY_CLASS<br>ENTITY_TYPE<br>FILE<br>INPUT_INTERFACE<br>OUTPUT_INTERFACE<br>R_NET<br>SUBNET<br>SYNONYM<br>VALIDATION_PATH |
| Relation-names: | EQUATES |
| Attribute-names: | TYPE |

Table 8.1 RSL Dependency of REVS Functions

| RSL CONCEPT/KEYWORD \ DEPENDENT REVS FUNCTION | RSL TRANSLATOR (RSL,RSLXTND) | INTERACTIVE R-NET GENERATION (RNETGEN) | DATA EXTRACTOR (RADX) | REQUIREMENTS ANALYZER (RADX) | SIMULATOR GENERATION (SIMGEN) |
|---|---|---|---|---|---|
| ELEMENT-TYPES | | | | | |
| SYNONYM | X | X | | | X |
| ALPHA | X | X | | X | X |
| EVENT | | X | | X | X |
| R_NET | X | X | X | X | X |
| SUBNET | X | X | X | X | X |
| DATA | X | X | | X | X |
| ENTITY_CLASS | X | X | | X | X |
| ENTITY_TYPE | X | X | | X | X |
| FILE | X | X | | X | X |
| INPUT_INTERFACE | X | X | | X | X |
| MESSAGE | | | | X | X |
| OUTPUT_INTERFACE | X | X | | X | X |
| VALIDATION_PATH | X | X | X | X | |
| VALIDATION_POINT | | X | | X | X |
| RELATIONS | | | | | |
| ASSOCIATES | | | | X | X |
| COMPOSES | | | | X | X |
| CONNECTS | | | | X | X |
| CONTAINS | | | | X | X |
| CREATES | | | | X | X |
| DELAYS | | | | X | X |
| DESTROYS | | | | X | X |
| ENABLES | | | | X | X |
| EQUATES | X | X | | | |
| FORMS | | | | X | X |
| INCLUDES | | | | X | X |
| INPUTS | | | | X | X |
| MAKES | | | | X | X |
| ORDERS | | | | X | X |
| OUTPUTS | | | | X | X |
| PASSES | | | | X | X |
| SETS | | | | X | X |
| ATTRIBUTES | | | | | |
| BETA | | | | | X |
| GAMMA | | | | | X |
| INITIAL_VALUE | | | | X | X |
| LOCALITY | | | | X | X |
| RANGE | | | | | X |
| TYPE | X | X | | | X |
| USE | | | | X | |
| STRUCTURE | X | X | X | X | X |

## 8.3 RNETGEN CHANGE CONSIDERATIONS

The RNETGEN software is totally dependent upon the structure concepts as documented in the REVS Users Manual. It will support the three structure types, R_NET, SUBNET, and VALIDATION_PATH, if their corresponding RSL element types exist in the ASSM. The following node types can be entered via RNETGEN if their corresponding RSL element types already exist in the ASSM.

| Node Types | RSL Element Type |
|---|---|
| INPUT | INPUT_INTERFACE |
| OUTPUT | OUTPUT_INTERFACE |
| ALPHA | ALPHA |
| OR | DATA, ENTITY_CLASS |
| FOR EACH | FILE, ENTITY_TYPE, ENTITY_CLASS |
| EVENT | EVENT |
| SELECT | ENTITY_TYPE,ENTITY_CLASS |
| SUBNET | SUBNET |
| VALPT | VALIDATION_POINT |
| AND, TERMINAL, RETURN | N/A |

There are other RSL definitions which are expected to have been previously entered in the ASSM but which, if absent, would not have any significant impact on RNETGEN. These are the relationship EQUATES, the element type SYNONYM, and the attribute TYPE.

## 8.4 RADX CHANGE CONSIDERATIONS

The data extraction portion of the RADX function is dependent on the structures of an R_NET, SUBNET, and VALIDATION_PATH. Should any changes be made to the types of nodes that appear on these structures or to the types of elements associated with the nodes, an assessment must be made of RADX to determine the effect of the change. There is also a dependency on the manner that named attribute-values are defined in RSL and processed by the RSL Translator.

The static analysis portion of RADX is highly dependent on the keywords used in RSL and also on the meaning of the keywords. The following lists them according to RSL primitives.

Element Types

| | |
|---|---|
| ALPHA | MESSAGE |
| DATA | OUTPUT_INTERFACE |
| ENTITY_CLASS | R_NET |
| ENTITY_TYPE | SUBNET |
| EVENT | VALIDATION_PATH |
| FILE | VALIDATION_POINT |
| INPUT_INTERFACE | |

Relations

| | |
|---|---|
| ASSOCIATES | INCLUDES |
| COMPOSES | INPUTS |
| CONNECTS | MAKES |
| CONTAINS | ORDERS |
| CREATES | OUTPUTS |
| DELAYS | PASSES |
| DESTROYS | SELECTS |
| ENABLES | SETS |
| FORMS | |

Attributes

INITIAL_VALUE
LOCALITY
USE

In addition to the explicitly defined RSL relations, RADX relies on the implicit relations REFERS and REFERRED. A REFERS relation exists between an element which has a structure and an element that is associated with a node on the structure. The REFERRED relation is the complement of REFERS.

## 8.5 SIMGEN CHANGE CONSIDERATIONS

The SIMGEN function is dependent on several RSL concepts and keywords, on the structure of R_NETs and SUBNETs, and on the syntax of special RSL statements which can appear in ALPHA models. The keywords for these special

statements are: CREATE, DESTROY, SELECT, FOR, and ENDFOREACH; statement syntax is defined in Section 3.5.4.

The concepts and keywords that SIMGEN requires are summarized below by the SIMGEN module which needs them.

<u>EVENT TRANSLATION</u>

CONNECTS
DATA
DELAYS
ENABLES
EVENT
INPUT_INTERFACE
OUTPUT_INTERFACE
R_NET
SUBSYSTEM

<u>ALPHA TRANSLATION</u>

ALPHA
BETA
CREATES
DATA
DESTROYS
ENTITY_CLASS
ENTITY_TYPE
FILE
FORMS
GAMMA
INPUTS
MESSAGE
OUTPUTS
RECORD_FOUND
SETS

R_NET/SUBNET TRANSLATION

ALPHA

DATA

ENTITY_CLASS

EVENT

INPUT_INTERFACE

OUTPUT_INTERFACE

R_NET

SUBNET

VALIDATION_POINT

VALIDATION TRANSLATION

CONTAINS

DATA

FILE

RECORDS

VALIDATION_POINT

DATA TRANSLATION

ASSOCIATES

COMPOSES

CONTAINS

DATA

ENTITY_CLASS

ENTITY_TYPE

FILE

INCLUDES

INITIAL_VALUE

INPUT_INTERFACE

LOCALITY

MAKES

MESSAGE

ORDERS

OUTPUT_INTERFACE

PASSES

RANGE

TYPE

PERFORMANCE_REQUIREMENT TRANSLATION

CONSTRAINS

FILE

PERFORMANCE_REQUIREMENT

TEST

VALIDATION_PATH

VALIDATION_POINT

## 8.6 SIMXQT CHANGE CONSIDERATIONS

The SIMXQT function is currently independent of RSL concepts and keywords.

## 8.7 SIMDA CHANGE CONSIDERATIONS

The SIMDA function is only dependent upon the RSL concepts and keyword PERFORMANCE_REQUIREMENT.

## 9.0 REFERENCES

1. "Software Design Specification - SREP Methodology", TRW Report No. 27332-6921-014, 1 October 1975.

2. "Process Design Methodology Design System Specification", Volumes I-V, Texas Instruments Incorporated, Document No. H750502-1B, September 1976.

3. "REVS Users Manual", SREP Final Report - Volume II, TRW Report No. 27332-6921-026, 1 August 1977.

4. "ASC Job Specification Language Reference Manual", Texas Instruments Incorporated, Document No. 930038.

5. "ASC Job Stream Utilities", Texas Instruments Incorporated, Document No. 930064.

6. "ASC FORTRAN Reference Manual", Texas Instruments Incorporated, Document No. 930044.

7. "ASC Linkage Editor User's Guide", Texas Instruments Incorporated, Document No. 930057.

8. "ASC Supervisor Service Calls", Texas Instruments Incorporated, Document No. 930035.

9. "ASC Source Management System User's Guide", Texas Instruments Incorporated, Document No. 931485.

10. "ASC Card Image File Editor User's Guide", Texas Instruments Incorporated, Document No. 930032.

11. "ASC Partitioned Direct Secondary Access Method Utilities", Texas Instruments Incorporated, Document No. 931487.

12. "BMDATC Data Processing Standards, Volume III, ARC Data Processing User's Guide", BMD Advanced Technology Center, Report No. TM-HU-212/003/00, July 1976.

13. "Interactive Graphics System Basic Functional Routines User's Manual", System Development Corporation, Report No. TM-HU-143/003/01, 28 June 1974.

14. "Interactive Graphics System Keyboard/Trackball Routines User's Manual", System Development Corporation, Report No. TM-HU-143/006/00, 20 December 1973 (Preliminary Draft).

15. E. A. Hershey III, "A Data Base Management System for PSA Based on DBTG 71", ISDOS Working Paper No. 88, University of Michigan, Department of Industrial and Operations Engineering, September 1973.

16. O. Lecarme and G. V. Bochmann, "A (Truly) Usable and Portable Translator Writing System", in: Rosenfeld, J. L. (ed.), Information Processing 74, Amsterdam, North-Holland, 1974.

17. O. Lecarme and G. V. Bochmann, "A Compiler Writing System User's Manual", University de Montreal, Department d' Informatique, Document de Travail # 57, December 1974.

18. "BMDATC Data Processing Standards, Volume I, Research and Development Documentation", BMD Advanced Technology Center, Report No. TM-HU-212/001/00, January 1976.

19. O. Lecarme, "A Compiler Writing System Installation Manual", University de Montreal, Department d' Informatique, Document de Travail # 58, January 1975.

20. "Scope 2.1 Reference Manual", Control Data Corporation, Publication No. 60342600.

21. "FORTRAN Extended Reference Manual", Control Data Corporation, Publication No. 60305600.

22. "Update Reference Manual", Control Data Corporation, Publication No. 60342500.

23. "Loader Reference Manual", Control Data Corporation, Publication No. 60344200.

24. "Compass Reference Manual", Control Data Corporation, Publication No. 60360900.

25. K. Jensen and N. Wirth, "PASCAL: User Manual and Report", Second Edition, Springer-Verlag New York Inc., New York, N.Y., 1975.

17. O. Lecarme and G. V. Bochmann, "A Compiler Writing System User's Manual", University de Montreal, Department d' Informatique, Document de Travail # 57, December 1974.

18. "BMDATC Data Processing Standards, Volume I, Research and Development Documentation", BMD Advanced Technology Center, Report No. TM-HU-212/001/00, January 1976.

19. O. Lecarme, "A Compiler Writing System Installation Manual", University de Montreal, Department d' Informatique, Document de Travail # 58, January 1975.

APPENDIX A

RSL TRANSLATOR ERROR MNEMONICS

| ERROR NO. | MNEMONIC | ERROR NO. | MNEMONIC | ERROR NO. | MNEMONIC | ERROR NO. | MNEMONIC | ERROR NO. | MNEMONIC |
|---|---|---|---|---|---|---|---|---|---|
| 0 | - | 411 | ERDELSENT | 438 | ERILLCOMP | 465 | EROBJINST | 492 | ERDATENCL |
| 1 | PILEDEBORDEE | 412 | not used | 439 | ERILLREM | 466 | ERONOMIX | 493 | ERNOTENUM |
| 2-4 | not used | 413 | ERDUPAINST | 440 | ERILLRNM | 467 | ERRELDEF | 494 | ERNOTFNTYPE |
| 5 | CARILLEGAL | 414 | ERDUPATTR | 441 | ERILLROBJ | 468 | ERRELSENT | 495 | EREVCRENT |
| 6 | GROSENTIER | 415 | ERDUPELTYP | 442 | ERILLRSUB | 469 | ERRETRNET | 496 | EREXTMODE |
| 7 | PRECISIONREEL | 416 | ERDUPETLM | 443 | ERILLTYPE | 470 | ERRINST | 497 | ERCNTRLMODE |
| 8 | GROSREEL | 417 | ERDUPFLAG | 444 | ERINVATTR | 471 | ERROBELT | 498 | ERDUPPERM |
| 9-10 | not used | 418 | ERDUPORD | 445 | ERINVELEM | 472 | ERRSBELT | 499 | ERNOPERM |
| 11 | TROPDERREURS | 419 | ERDUPREL | 446 | ERINVELT | 473 | ERSTNONODE | 500 | ERNOTEXT |
| 12-199 | - | 420 | not used | 447 | ERINVREL | 474 | ERSTNOTERM | 501 | ERLASTCNTRL |
| 200 | ERRECOVERY | 421 | ERDUPROLST | 448 | ERNAMOFLO | 475 | ERSTRUCT | 502 | ERNOCNTRL |
| 201 | ERCOMEXP | 422 | ERDUPSTR | 449 | ERNAMUSE | 476 | ERSUBJINST | 503 | ERNOBRANCH |
| 202 | ERCOMNAL | 423 | ERDUPVAL | 450 | ERNDFLTERM | 477 | ERSYNUSE | 504 | ERNULLBR |
| 203 | ERPERMISS | 424 | ERELEMDEF | 451 | ERNEWLOST | 478 | ERTNTMIX | 505 | ERDUPCNAME |
| 204 | ERDUPRINST | 425 | ERELEMSENT | 452 | ERNIMPL | 479 | ERUNDNAME | 506 | ERCONDBLNK |
| 205-399 | not used | 426 | ERELMELT | 453 | ERNOATTR | 480 | ERVALNODE | 507-599 | not used |
| 400 | ERAINST | 427 | ERELNODE | 454 | ERNOAVAL | 481 | ERVALPATH | 600 | ERILLNAM |
| 401 | ERAPPELTYP | 428 | ERELTDEF | 455 | ERNOBJLIST | 482 | ERVALSPEC | 601 | ERINITIAL |
| 402 | ERASSNODE | 429 | ERELTNPUSE | 456 | ERNODENAL | 483 | ERVNDNAL | 602 | ERREVS |
| 403 | ERATTELT | 430 | ERELTSENT | 457 | ERNONPFLAG | 484 | ERVPNONODE | 603 | ERASSM |
| 404 | ERATTRDEF | 431 | ERRECNOFILE | 458 | ERNOPATH | 485 | ERZERORD | 604 | ERCONNEES |
| 405 | ERATTRSENT | 432 | ERFEBODY | 459 | ERNORINST | 486 | ERENTITY | 605 | ERRSLDICT |
| 406 | ERBIGORD | 433 | ERFILENT | 460 | ERNOSTR | 487 | ERDUPRETYPE | 606 | ERNORESWORD |
| 407 | ERCONDEXPR | 434 | ERFOLDO | 461 | ERNOTYPE | 488 | ERIISUBN | | |
| 408 | ERCRDEFHD | 435 | ERIIFBGBR | 462 | ERNOSECT | 489 | ERNORETN | | |
| 409 | ERDATANAME | 436 | ERIIFLNODE | 463 | ERNOTLIST | 490 | ERMULRETN | | |
| 410 | ERULGEXPR | 437 | ERILLAVAL | 464 | ERNOVAL | 491 | EROIFRLTYPE | | |

# APPENDIX B

# REVS INSTALLATION AND MAINTENANCE ON CDC 7600 AT THE ARC

## INSTALLATION AND MAINTENANCE PROCEDURES

This section describes how to install and maintain the many different components of REVS from the Software Deliverable File (SDF) delivered on tape to the ARC. Included are deck setups and corresponding descriptions for performing the following procedures:

1) Create an UPDATE source file from the delivered SDF tape.
2) Create the RSL translator and its required external input files, RSLDICT and DONNEES.
3) Construct the DBCS library which is subsequently used to create new load modules of REVS, VVDBLDR, and VV post-processor.
4) Construct a new REVS load module.
5) Construct the VV library (VVLIBE) and the VV database builder (VVDBLDR).
6) Create null ASSM and VV database.
7) Create the RISF file.
8) Construct the ASC JSL Emulators.
9) Construct the REVSLIB library.
10) Create a nominal ASSM database.

There is a job dependency associated with the jobs listed above as presented by the Job Dependency Chart in Figure B-1. The numbers listed along the X and Y axis in the chart correspond to the jobs as listed above. It should be noted that job (1) above must be run prior to running any of the other jobs in the list.

The Software Deliverable File contains source in UPDATE format for all REVS components required for installing REVS from scratch as follows:

- Lecarme-Bochmann Compiler Writing System (LB-CWS)
- Data Base Control System (DBCS)
- REVS
- PASCAL Compiler, Library, Utilities
- RISF

| JOBS TO RUN \ DEPENDENT JOBS | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | |
| 2 | X | | | | | | | | | |
| 3 | X | | | | | | | | | |
| 4 | X | X | X | | | | | | | |
| 5 | X | X | X | | | | | | | |
| 6 | X | X | X | | | | | | | |
| 7 | X | | | | | | | | | |
| 8 | X | | | | | | | | | |
| 9 | X | X | X | | X | | | X | | |
| 10 | X | X | X | X | X | X | X | X | X | |

Figure B-1 Job Dependency Chart

- RSL Nucleus
- DDL for ASSM and VV data bases (XXDDL, VVDDL)
- SEGLOAD Overlay Commands (OVERLAY)

## CREATE UPDATE SOURCE FILE OF SDF

The job setup shown in Figure B-2 creates an UPDATE source file using the Software Deliverable File (SDF) as delivered on tape to the ARC. This job must be run prior to running any of the other jobs identified in the previous section since all other jobs access the permanent file generated by this job. The job produces a complete UPDATE listing of the SDF for future reference.

```
JOBNO,STMFZ,T77,NT1.    CREATE SDF FILE            YOUR NAME
REQUEST,NEWPL,*PF.
FILE,OLDPL,RT=W,BT=I.
STAGE,OLDPL,VSN=SDFTAPENO.
COMMENT.      LIST SDF
UPDATE,L=F,C=0,F,N.
COMMENT.      CATALOG SDF UNDER YOURID
CATALOG,NEWPL,REVSDELIVERPL,ID=YOURID.
              7/8/9  CARD
*LT 500000
              6/7/8/9  CARD
```

Figure B-2 Deck Setup to Create SDF UPDATE File

## CREATE RSL TRANSLATOR

The RSL translator is constructed using the Lecarme-Bochmann Compiler Writing System (L-B CWS) as described in Section 7.3.1 of this manual. Two files are created during this process (see Section 7.2.2) which are subsequently used as input files when executing the RSL translation function within REVS. These files are named RSLDICT and DONNEES. Figure B-3 provides the deck setup used to construct a new RSL translator and its corresponding RSLDICT and DONNEES files. The setup also provides for updating the REVS source code of the SDF with the newly constructed translator.

```
JOBNO,STMFZ,T77.          CREATE RSL TRANSLATOR        YOUR NAME
REQUEST,DONNEES,*PF.
REQUEST,RSLDICT,*PF.
COMMENT.        GET SDF FILE
ATTACH,PL,REVSDELIVERPL,ID=YOURID.
COMMENT.        UPDATE CWS INPUT/PROGRAM DECKS
UPDATE,F,P=PL,N=OLDPL,C=0.
RETURN,PL.
COMMENT.        UPDATED CWS PL IS ON FILE OLDPL
COMMENT.        OBTAIN CWS INPUT FILES FROM OLDPL
UPDATE,Q,D,8,C=RSLSYN,L=1.
UPDATE,Q,D,8,C=GLOBAUX,L=1.
UPDATE,Q,D,8,C=NOYALEX,L=1.
UPDATE,Q,D,8,C=NOYASYN,L=1.
UPDATE,Q,D,8,C=ERREURS,L=1.
COMMENT.        OBTAIN CWS RPOGRAM FILES FROM OLDPL
UPDATE,Q,D,8,C=SEMA,L=1.
UPDATE,Q,D,8,C=SYNT,L=1.
UPDATE,Q,D,8,C=LEXI,L=1.
UPDATE,Q,D,8,C=COMP,L=1.
RETURN,OLDPL.
RFL,60000.
COMMENT.        ATTACH PASCAL COMPILER/LIBRARY
ATTACH,PLIB,PASCALLIBRARY,ID=SYSID.
LIBRARY,PLIB.
COMMENT.        COMPILE AND EXECUTE SEMAGEN PROGRAM
PASCAL,SEMA,OUTPUT,SEMAGEN.
SEMAGEN,RSLSYN.
COMMENT.        COMPILE AND EXECUTE SYNTGEN PROGRAM
PASCAL,SYNT,OUTPUT,SYNTGEN.
SYNTGEN.
COMMENT.        COMPILE AND EXECUTE LEXIGEN PROGRAM
PASCAL,LEXI,OUTPUT,LEXIGEN.
LEXIGEN.
COMMENT.        COMPILE AND EXECUTE COMPGEN PROGRAM
PASCAL,COMP,OUTPUT,COMPGEN.
COMPGEN.
COMMENT.        RSL TRANSLATOR SOURCE IS ON FILE COMPILE
REWIND,COMPILE.
REDUCE,S.
COMMENT.        CATALOG NEW DONNEES AND RSLDICT FILES
CATALOG,DONNEES,YOURNAMEFORDONNEES,ID=YOURID.
CATALOG,RSLDICT,YOURNAMEFORRSLDICT,ID=YOURID.
COMMENT.        GET SDF FILE
ATTACH,OLDPL,REVSDELIVERPL,ID=YOURID.
REQUEST,NEW,*PF.
COMMENT.        PURGE EXISTING COMDECK TTRSL
UPDATE,N=OLD,L=A12,C=0.
COMMENT.        ADD NEW COMDECK TTRSL
UPDATE,E,P=OLD,N=NEW,L=A12,C=0.
COMMENT.        CATALOG REVS CDC PL WITH UPDATED RSL TRANSLATOR
CATALOG,NEW,REVSDELIVERPL,ID=YOURID.
```

Figure B-3 Deck Setup for Creating the RSL Translator

```
                         7/8/9 CARD
*IDENT YOURMODS
*/ PUT ANY MODS TO CWS PROGRAMS OR INPUT FILES HERE
*IDENT LMINUS
*/ THESE MODS TURN OFF THE PASCAL COMPILE LISTING FOR ALL CWS PROGRAMS
*I COMPGEN.1
(*$L- *)
*I LEXIGEN.1
(*$L- *)
*I PROGGEN.1
(*$L- *)
*I SEMAGEN.1
(*$L- *)
*I SYNTGEN.1
(*$L- *)
                         7/8/9 CARD
*/ RSLSYN CONTAINS THE SYNTACTIC DEFINITION OF RSL
*COMPILE RSLSYN
                         7/8/9 CARD
*/ RSLGLO CONTAINS THE SEMANTIC ROUTINES FOR RSL
*/ THIS FILE IS KNOWN AS GLOBAUX BY THE CWS
*COMPILE RSLGLO
                         7/8/9 CARD
*/ NOYALEX CONTAINS THE STANDARD PARTS OF THE LEXICAL ANALYZER
*COMPILE NOYALEX
                         7/8/9 CARD
*/ NOYASYN CONTAINS THE STANDARD PARTS OF THE SYNTACTIC ANALYZER
*COMPILE NOYASYN
                         7/8/9 CARD
*/ ERREURS CONTAINS THE STANDARD PARTS OF THE ERROR TREATMENT
*COMPILE ERREURS
                         7/8/9 CARD
*/ SEMAGEN READS THE RSLSYN FILE
*COMPILE SEMAGEN
                         7/8/9 CARD
*/ SYNTGEN GENERATES THE SYNTACTIC ANALYZER FOR RSL
*COMPILE SYNTGEN
                         7/8/9 CARD
*/ LEXIGEN GENERATES THE LEXICAL ANALYZER FOR RSL
*COMPILE LEXIGEN
                         7/8/9 CARD
*/ COMPGEN COMBINES ALL PARTS OF THE GENERATED RSL TRANSLATOR
*COMPILE COMPGEN
                         7/8/9 CARD
*/ PURGE THE OLD COMDECK TTRSL AND THE IDENT TTRSL
*PURGE TTRSL
*PURDECK TTRSL
                         7/8/9 CARD
*/ READ IN THE NEW COMDECK TTRSL
*ADDFILE INPUT,TESTER
*COMDECK TTRSL
*READ COMPILE
                         7/8/9 CARD
                         6/7/8/9 CARD
```

Figure B-3 Deck Setup for Creating the RSL Translator (Continued)

## CONSTRUCT DBCS LIBRARY

The Data Base Control System (DBCS) used to maintain the ASSM (see Section 7.3.2) must be constructed in library format to be used in constructing load modules of REVS, VV data base builder, and Simulator postprocessor. One of several different versions of DBCS can be generated by means of DEFINE parameters selected during the UPDATE process. These DEFINE parameters provide for data base pages to reside either in LCM, SCM, or both and to invoke optimization capabilities for improving cp run time. Following is a list of the allowable combinations of DEFINE parameters and a description of the resulting DBCS:

| DEFINE Parameters | DBCS Description |
|---|---|
| (1) LCM, OPTIMIZE | - optimized version of DBCS with data base pages residing in LCM only (NOMINAL) |
| (2) LCMSCM, OPTIMIZE | - optimized version of DBCS with data base pages residing both in LCM and SCM. |
| (3) OPTIMIZE | - optimized version of DBCS with data base pages residing in SCM only. |
| (4) LCM | - unoptimized version of DBCS with data base pages residing in LCM only. |
| (5) LCMSCM | - unoptimized version of DBCS with data base pages residing both in LCM and SCM. |
| (6) no parameters specified | - unoptimized version of DBCS with data base pages residing in SCM only (stock version). |

Figure B-4 presents the deck setup for creating the nominal DBCS library, i.e., optimized version with data base pages residing in LCM only. The setup also provides for constructing new load modules for the data base utility programs, DDLA, DBIN, and DBUT1. These utilities are used in defining, initializing, and dumping data bases (see Section 7.3.2).

```
JOBNO,STMFZ,T77.        CREATE DBCS LIBRARY        YOUR NAME
REQUEST,NEWDBCS,*PF.
COMMENT.        GET SDF FILE
ATTACH,OLDPL,REVSDELIVERPL,ID=YOURID.
UPDATE,Q,L=1A.
COMMENT         COMPILE DBCS SOURCE
FTN,I.
COMMENT.        CREATE TEMPORARY LIBRARY
LIBEDT.
LIBRARY,DBCS.
COMMENT.        CONSTRUCT DBIN LOAD MODULE
LIBLOAD,DBCS,DBIN.
SLOAD,DBCS/NOMINL,NOMINL.
SLOAD,DBCS/PIC2,PIC2.
NOGOF1,DBIN.
COMMENT.        CONSTRUCT DBUTI LOAD MODULE
LIBLOAD,DBCS,DBUTI.
SLOAD,DBCS/NOMINL,NOMINL.
SLOAD,DBCS/PIC2,PIC2.
NOGO,F2,DBUTI.
COMMENT.        CONSTRUCT DDLA LOAD MODULE
LIBLOAD,DBCS,DDLA.
SLOAD,DBCS/NOMINL,NOMINL.
NOGO,F3,DDLA.
COMMENT.        ADD LOAD MODULES TO DBCS LIBRARY
LIBEDT.
CATALOG,NEWDBCS,REVSDBCSLIBE,ID=YOURID.
                7/8/9  CARD
*ID TEMP
*DF LCM,OPTIMIZE
*C MISS$$$,DBUTI.CTOI,BLKFIX,BLKPR02.CLOSEMS,INITS,LLEFT,LLEFTI
*C LRIGHT,LRIGHI,LSMOVE,SSMOVE,LSCOMP,BLKPR32,BLKPR64
*C BLKPR128,BLKPR100,BLKPR200,LWPAGE,LCM2CM,CM2LCM
                7/8/9  CARD
LIBRARY(DBCS,NEW=4000)
REWIND(LGO)
ADD(*,LGO)
FINISH.
ENDRUN.
                7/8/9  CARD
LIBRARY(DBCS,OLD)
DELETE(DBIN)
DELETE(DBUTI)
DELETE(DDLA)
DELETE(DBINSB)
DELETE(DDLASB)
DELETE(DBUTSB)
REWIND(F1)
REWIND(F2)
REWIND(F3)
ADD(*,F1)
ADD(*,F2)
ADD(*,F3)
FINISH.
LIBRARY(NEWDBCS,NEW=4000)
OLDLIB(DBCS)
FINISH.
ENDRUN.
                6/7/8/9     CARD
```

Figure B-4 Deck Setup for Constructing DBCS Library

CONSTRUCT REVS LOAD MODULE

The deck setup to compile the entire REVS program and construct a new REVS load module is presented in Figure B-5. The same deck setup may be used for compilation of selective REVS functions to create updated versions of the load module. This is accomplished by inserting an ATTACH card at the appropriate place in the deck (see COMMENT cards in Figure B-5) for a previously saved version of REVSLGO. Inputs to the second UPDATE must be provided accordingly. The following table lists available DEFINE parameters for selective compilation of REVS.

| *DEFINE NAME | SOURCE CODE TO BE COMPILED |
|---|---|
| AAPROC | All ASSM ACCESS Procedures |
| CCNET | All CALCOMP plotting procedures |
| QQRADX | QQRADX procedure only |
| RADX | RADX and all lower level procedures |
| RNETGEN | RNETGEN and all lower level procedures |
| RSL | RSL procedure only |
| RSLXTND | RSLXTND procedure only |
| TTRSL | TTRSL and all lower level procedures |
| SIMDA | SIMDA and all lower level procedures |
| SIMGEN | SIMGEN and all lower level procedures |
| SIMXQT | SIMXQT and all lower level procedures |
| TESTER | TESTER and all lower level procedures |
| XXREVS | All XX procedures |

To compile all of REVS, the user must provide all of the *DEFINE names given above.

```
JOBNO,STMFZ,T777.       CREATE REVS LOAD MODULE    YOUR NAME
COMMENT.      GET PASCAL LIBRARY FROM SYSTEM
ATTACH,PLIB,PASCALLIBRARY,ID=SYSID.
COMMENT.      GET DBCS LIBRARY
ATTACH,DBCS,REVSDBCSLIBE,ID=YOURID.
COMMENT.      GET ANAGRAPH LIBRARY
ATTACH,ARCLIB,ID=PRDLIB.
COMMENT.      GET SDF FILE
ATTACH,OLDPL,REVSDELIVERPL,ID=YOURID.
LIBRARY,DBCS,PLIB,ARCLIB.
COMMENT.      GET AND COMPILE FORTRAN SOURCE
UPDATE,Q,L=1A.
FTN,I,L=0,B=REVSFTN.
COMMENT.      GET AND COMPILE PASCAL SOURCE
RETURN,COMPILE.
UPDATE,Q,L=1A.
RFL,110000.
PASCAL,COMPILE,ROUT.
REDUCE.
COMMENT.      DISPLAY PASCAL COMPILE ERRORS, IF ANY
FINERR,ROUT.
COMMENT.      INSERT CARD HERE TO ATTACH PREVIOUSLY CATALOGGED
COMMENT.      VERSION OF REVSLGO, IF APPLICABLE
REQUEST,NEWLGO,*PF.
REWIND,LGO,REVSLGO.
COMMENT.      MERGE NEW LGO WITH OLD LGO, IF APPLICABLE
MERGEL,REVSLGO,LGO,NEWLGO.
RETURN,REVSLGO.
COMMENT.      SAVE NEWLGO FOR FUTURE USE
CATALOG,NEWLGO,REVSLGO,ID=YOURID.
COMMENT.      COPY PIC200 TO SEQUENTIAL FILE
LIBEDT.
REQUEST,REVSABS,*PF.
COMMENT.      GET SEQLOAD OVERLAY COMMANDS
UPDATE,Q,L=1A,C=OVLYCMD.
COMMENT.      CONSTRUCT REVS LOAD MODULE
SEQLOAD,I=OVLYCMD,B=REVSABS.
LDSET,ERR=NONE,MAP=/REVSMAP,PRESET=ZERO.
LOAD,REVSFTN.
LOAD,DBCSPIC.
LOAD,NEWLGO.
NOGO.
CATALOG,REVSABS,REVSABS,ID=YOURID.
              7/8/9  CARD
*ID TEMP
*C REVSFTN
              7/8/9  CARD
*ID TEMP
*DF AAPROC,CCNET,QQRADX,RADX,RNETGEN,RSL,TTRSL,RSLXTND
*DF SIMDA,SIMXQT,SIMGEN,TESTER,XXREVS
*/  PUT ANY MODS TO REVS FUNCTIONS HERE
*C OPTIONS,REVS
              7/8/9  CARD
LIBRARY(DBCS,OLD)
PCOPY(PIC200,DBCSPIC)
FINISH.
ENDRUN.
              7/8/9  CARD
*ID TEMP
*C OVERLAY
              6/7/8/9     CARD
```

Figure B-5 Deck Setup for Constructing REVS Load Module

## CONSTRUCT VV LIBRARY

The deck setup presented in Figure B-6 is used to build a new VV library (VVLIBE). The generated library is then used to construct a load module of the VV data base builder (VVDBLDR). The VV library is also later used in constructing a load module of the VV post-processor. The latter is an automated process which occurs during the execution of REVS.

```
JOBNO,STMFZ,T77.          CREATE VVLIBE,VVDBLDR        YOUR NAME
REQUEST,VVDBLDR,*PF.
REQUEST,VVLIBE,*PF.
COMMENT.        GET SDF FILE
ATTACH,OLDPL,REVSDELIVERPL,ID=YOURID.
COMMENT.        GET AND COMPILE FORTRAN SOURCE FOR VVLIBE
UPDATE,Q,L=1A.
FTN,I,L=0,B=VVFTN.
RETURN,COMPILE.
COMMENT.        GET PASCAL AND DBCS LIBRARIES
ATTACH,PLIB,PASCALLIBRARY,ID=SYSID.
ATTACH,DBCS,REVSDBCSLIBE,ID=YOURID.
LIBRARY,PLIB.
COMMENT.        GET AND COMPILE PASCAL SOURCE FOR VVLIBE
UPDATE,Q,L=1A.
PASCAL,COMPILE,VVOUT.
COMMENT. DISPLAY PASCAL ERRORS, IF ANY
RESET,VVOUT.
FINERR,VVOUT.
COMMENT.        CONSTRUCT VV LIBRARY
LIBEDT.
RETURN,COMPILE,LGO,VVOUT.
COMMENT.        GET AND COMPILE VVDBLDR
UPDATE,Q,L=1A.
PASCAL,COMPILE,VVOUT.
COMMENT.        DISPLAY PASCAL ERRORS, IF ANY
RESET,VVOUT.
FINERR,VVOUT.
COMMENT.        CONSTRUCT VVDBLDR LOAD MODULE
LIBRARY,VVLIBE,DBCS,PLIB.
LIBLOAD,VVLIBE,FVVBLDR.
SLOAD,VVLIBE/NOMINL,NOMINL.
SLOAD,DBCS/PIC32,PIC32.
LOAD,LGO.
NOGO,VVDBLDR.
CATALOG,VVDBLDR,VVDBLDR,ID=YOURID.
CATALOG,VVLIBE,VVLIBE,ID=YOURID.
                7/8/9  CARD
*ID TEMP
*C VVFTN
                7/8/9  CARD
*ID TEMP
*C VVEXEC
                7/8/9  CARD
LIBRARY(VVLIBE,NEW=2000)
REWIND(VVFTN)
ADD(*,VVFTN)
REWIND(LGO)
ADD(*,LGO)
FINISH.
ENDRUN.
                7/8/9  CARD
*ID TEMP
*C VVDBLDR
                6/7/8/9     CARD
```

Figure B-6 Deck Setup for Creating VV Library

## CREATE NULL DATA BASES

The deck setup shown in Figure B-7 is used to create both a null ASSM data base and a null VV recording data base (see Section 7.3.2). These data bases and their corresponding data base tables are catalogued for use by subsequent installation and execution jobs of REVS.

```
JOBNO,STMFZ,T77.    CREATE ASSM AND VV DATABASES  YOUR NAME
REQUEST,TAPE2,*PF.
REQUEST,TAPE3,*PF.
REQUEST,TAPE10,*PF.
REQUEST,TAPE11,*PF.
COMMENT.       GET SDF FILE
ATTACH,OLDPL,REVSDELIVERPL,ID=YOURID.
COMMENT.       GET DBCS UTILITIES
ATTACH,DBCS,REVSDBCSLIBE,ID=YOURID.
LIBRARY,DBCS.
COMMENT.       GET DDL FOR ASSM DATABASE
UPDATE,Q,L=1A,D,8,C=DDL.
DDLA,DDL.
DBIN.
CATALOG,TAPE2,ASSMDB,ID=YOURID.
CATALOG,TAPE3,ASSMDBT,ID=YOURID.
RETURN,DDL.
COMMENT.       GET DDL FOR VV DATABASE
UPDATE,Q,L=1A,D,8,C=DDL.
DDLA,DDL.,TAPE11.
DBIN,TAPE10,TAPE11.
CATALOG,TAPE10,VVDB,ID=YOURID.
CATALOG,TAPE11,VVDBT,ID=YOURID.
               7/8/9  CARD
*ID TEMP
*C XXDDL
               7/8/9  CARD
*ID TEMP
*C VVDDL
               6/7/8/9     CARD
```

Figure B-7  Deck Setup for Creating Null REVS Data Base

## CREATE RISF FILE

The deck setup given in Figure B-8 is used to generate the Requirements Independent Source File (see Section 7.2.3) which is used during the execution of the SIMGEN function in REVS.

```
JOBNO,STMFZ,T77.        CREATE RISF                    YOUR NAME
REQUEST,RISF,*PF.
COMMENT.        GET SDF FILE
ATTACH,OLDPL,REVSDELIVERPL,ID=YOURID.
UPDATE,Q,L=1A,D,8,C=RISF.
CATALOG,RISF,RISF,ID=YOURID.
                7/8/9  CARD
*ID TEMP
*C GGRISF
                6/7/8/9     CARD
```

Figure B-8  Deck Setup to Create RISF

CONSTRUCT JSL EMULATOR LOAD MODULE

The deck setup given in Figure B-9 is used to assemble and construct the load module for the JSL emulators. The generated load module is catalogued for subsequent inclusion in the REVSLIB library, see following paragraph.

```
JOBNO,STMFZ,T77.        CREATE JSL EMULATORS       YOUR NAME
COMMENT.       GET SDF FILE
ATTACH,OLDPL,REVSDELIVERPL,ID=YOURID.
COMMENT.       GET AND ASSEMBLE EMULATOR SOURCE
UPDATE,Q.
COMPASS,D,S=SYSTEXT,S=LDRTEXT,S=PFMTEXT,I.
COMMENT.       CONSTRUCT JSL EMULATORS LOAD MODULE
LOAD,LGO.
NOGO,EMULATR,REVSPRE,REVSXQT,SIMBUIL,SIMRUN,TESTRUN,SIMLOAD,SIMSAVE.
REQUEST,REVSMAC,*PF.
LIBEDT.
CATALOG,REVSMAC,ID=YOURID.
               7/8/9 CARD
*ID TEMP
*C REVSMAC
               7/8/9 CARD
LIBRARY(REVSMAC,NEW)
REWIND(EMULATR)
REPLACE(*,EMULATR)
FINISH.
               6/7/8/9  CARD
```

Figure B-9 Deck Setup for Constructing JSL Emulator Load Module

CONSTRUCT REVSLIB

The deck setup given in Figure B-10 provides for the construction of a library called REVSLIB which is a consolidation of the previously generated libraries, DBCS and VVLIBE. It will also contain the JSL emulators described in the previous paragraph. REVSLIB is subsequently used in the REVS execution deck setup as indicated in the following paragraph.

```
JOBNO,STMFZ,T77.        CREATE REVSLIB                YOUR NAME
REQUEST,REVSLIB,*PF.
COMMENT.        GET DBCS AND VVLIBE AND JSL EMULATORS
ATTACH,DBCS,REVSDBCSLIBE,ID=YOURID.
ATTACH,VVLIBE,VVLIBE,ID=YOURID.
ATTACH,REVSMAC,ID=YOURID.
COMMENT.        CONSTRUCT REVSLIB LIBRARY
LIBEDT.
CATALOG,REVSLIB,REVSLIB,ID=YOURID.
                7/8/9  CARD
LIBRARY(REVSLIB,NEW=4000)
OLDLIB(DBCS)
REPLACE(*,REVSMAC)
REPLACE(*,VVLIBE,LIB)
FINISH.
ENDRUN.
                6/7/8/9      CARD
```

Figure B-10 Deck Setup to Construct REVSLIB

## CREATE NOMINAL ASSM

The deck setup shown in Figure B-11 is the nominal deck setup for executing REVS, with the exception of the cards indicated. These additional cards were inserted in order to access a null ASSM, since the nominal ASSM (RSL nucleus) does not as yet, exist. RSL inputs for defining the ASSM nucleus are retrieved from the SDF file via an UPDATE step, as indicated. The resulting data base on file TAPE2 is catalogued for future REVS executions.

```
JOBNO,STMFZ,T77.        CREATE NOMINAL ASSM        YOUR NAME
ATTACH,REVSLIB,ID=YOURID.
COMMENT.        GET PASCAL LIBRARY
ATTACH,PLIB,PASCALLIBRARY,ID=SYSID.
LIBRARY,REVSLIB,PLIB.
COMMENT.        PERFORM REVSPREP MACRO
REVSPRE.
COMMENT.        GET NULL DATABASE. THE FOLLOWING FIVE CARDS
COMMENT.        SHOULD BE REMOVED FOR NOMINAL REVS EXECUTION.
RETURN,TAPE2.
REQUEST,TAPE2,*PF.
GETPF,TAPE2,ASSMDB,ID=YOURID.
ATTACH,OLDPL,REVSDELIVERPL,ID=YOURID.
UPDATE,Q,L=1A,D,8,C=NUCLEUS.
EXIT,U.
REVSXQT.
CATALOG,TAPE2,ASSMDBRSLNUCLEUS,ID=YOURID.
                7/8/9  CARD
*ID TEMP
*/  THIS INPUT SECTION ALONG WITH CORRESPONDING UPDATE
*/  COMMAND SHOULD BE REMOVED FOR NOMINAL REVS EXECUTION
*C NUCLEUS
                7/8/9  CARD
RSLXTND.
ADDFILE NUCLEUS.
STOP.
                6/7/8/9     CARD
```

Figure B-11 Deck Setup for Creating Nominal ASSM